Distributed Computing with Adaptive Heuristics

Revised version will appear in the *Proceedings of Innovations in Computer Science 2011*

Aaron D. Jaggard *
Dept. of Computer Science, Colgate University
DIMACS, Rutgers University
adj@dimacs.rutgers.edu

Michael Schapira†
Dept. of Computer Science
Yale University and UC Berkeley
michael.schapira@yale.edu

Rebecca N. Wright ‡
Dept. of Computer Science and DIMACS
Rutgers University
rebecca.wright@rutgers.edu

Abstract

We use ideas from distributed computing to study dynamic environments in which computational nodes, or decision makers, follow *adaptive heuristics* [16], *i.e.*, simple and unsophisticated rules of behavior, *e.g.*, repeatedly "best replying" to others' actions, and minimizing "regret", that have been extensively studied in game theory and economics. We explore when convergence of such simple dynamics to an equilibrium is guaranteed in asynchronous computational environments, where nodes can act at any time. Our research agenda, *distributed computing with adaptive heuristics*, lies on the borderline of computer science (including distributed computing and learning) and game theory (including game dynamics and adaptive heuristics). We exhibit a general non-termination result for a broad class of heuristics with bounded recall—that is, simple rules of behavior that depend only on recent history of interaction between nodes. We consider implications of our result across a wide variety of interesting and timely applications: game theory, circuit design, social networks, routing and congestion control. We also study the computational and communication complexity of asynchronous dynamics and present some basic observations regarding the effects of asynchrony on no-regret dynamics. We believe that our work opens a new avenue for research in both distributed computing and game theory.

*Supported in part by NSF grants 0751674 and 0753492.
†Supported by NSF grant 0331548.
‡Supported in part by NSF grant 0753061.

1 Introduction

Dynamic environments where computational nodes, or decision makers, repeatedly interact arise in a variety of settings, such as Internet protocols, large-scale markets, social networks, multi-processor computer architectures, and more. In many such settings, the prescribed behavior of the nodes is often simple, natural and myopic (that is, a heuristic or "rule of thumb"), and is also adaptive, in the sense that nodes constantly and autonomously react to others. These "adaptive heuristics"—a term coined in [16]—include simple behaviors, *e.g.*, repeatedly "best replying" to others' actions, and minimizing "regret", that have been extensively studied in game theory and economics.

Adaptive heuristics are simple and unsophisticated, often reflecting either the desire or necessity for computational nodes (whether humans or computers) to provide quick responses and have a limited computational burden. In many interesting contexts, these adaptive heuristics can, in the long run, move the global system in good directions and yield highly rational and sophisticated behavior, such as in game theory results demonstrating the convergence of best-response or no-regret dynamics to equilibrium points (see [16] and references therein).

However, these positive results for adaptive heuristics in game theory are, with but a few exceptions (see Section 2), based on the sometimes implicit and often unrealistic premise that nodes' actions are somehow synchronously coordinated. In many settings, where nodes can act at any time, this kind of synchrony is not available. It has long been known that asynchrony introduces substantial difficulties in distributed systems, as compared to synchrony [12], due to the "limitation imposed by local knowledge" [24]. There has been much work in distributed computing on identifying conditions that guarantee protocol termination in asynchronous computational environments. Over the past three decades, we have seen many results regarding the possibility/impossibility borderline for failure-resilient computation [11, 24]. In the classical results of that setting, the risk of non-termination stems from the possibility of failures of nodes or other components.

We seek to bring together these two areas to form a new research agenda on distributed computing with adaptive heuristics. Our aim is to draw ideas from distributed computing theory to investigate provable properties and possible worst-case system behavior of adaptive heuristics in asynchronous computational environments. We take the first steps of this research agenda. We show that a large and natural class of adaptive heuristics fail to provably converge to an equilibrium in an asynchronous setting, even if the nodes and communication channels are guaranteed to be failure-free. This has implications across a wide domain of applications: convergence of game dynamics to pure Nash equilibria; stabilization of asynchronous circuits; convergence to a stable routing tree of the Border Gateway Protocol, that handles Internet routing; and more. We also explore the impact of scheduling on convergence guarantees. We show that non-convergence is not inherent to adaptive heuristics, as some forms of regret minimization provably converge in asynchronous settings. In more detail, we make the following contributions:

General non-convergence result (Section 4). It is often desirable or necessary due to practical constraints that computational nodes' (*e.g.*, routers') behavior rely on limited memory and processing power. In such contexts, nodes' adaptive heuristics are often based on *bounded recall*—*i.e.*, depend solely on recent history of interaction with others—and can even be *historyless*—*i.e.*, nodes only react to other nodes' current actions). We exhibit a general impossibility result using a valency argument—a now-standard technique in distributed computing theory [11, 24]—to show that a broad class of bounded-recall adaptive heuristics cannot always converge to a stable state. More specifically, we show that, for a large family of such heuristics, simply the existence of two

"equilibrium points" implies that there is some execution that does not converge to any outcome even if nodes and communication channels are guaranteed not to fail. We also give evidence that our non-convergence result is essentially tight.

Implications across a wide variety of interesting and timely applications (Section 5). We apply our non-convergence result to a wide variety of interesting environments, namely convergence of game dynamics to pure Nash equilibria, stabilization of asynchronous circuits, diffusion of technologies in social networks, routing on the Internet, and congestion control protocols.

Implications for convergence of r-fairness and randomness (Section 6). We study the effects on convergence to a stable state of natural restrictions on the order of nodes' activations (i.e., the order in which nodes' have the opportunity to take steps), that have been extensively studied in distributed computing theory: (1) *r-fairness*, which is the guarantee that each node selects a new action at least once within every r consecutive time steps, for some pre-specified $r > 0$; and (2) randomized selection of the initial state of the system and the order of nodes' activations.

Communication and computational complexity of asynchronous dynamics (Section 7). We study the tractability of determining whether convergence to a stable state is guaranteed. We present two complementary hardness results that establish that, even for extremely restricted kinds of interactions, this feat is hard: (1) an exponential communication complexity lower bound; and (2) a computational complexity PSPACE-completeness result that, alongside its computational implications, implies that we cannot hope to have short witnesses of guaranteed asynchronous convergence (unless PSPACE $\subseteq$ NP).

Asynchronous no-regret dynamics (Section 8). We present some basic observations about the convergence properties of no-regret dynamics in our framework, that establish that, in contrast to other adaptive heuristics, regret minimization is quite robust to asynchrony.

Further discussion of a research agenda in distributed computing with adaptive heuristics (Section 9) We believe that this work has but scratched the surface in the exploration of the behavior of adaptive heuristics in asynchronous computational environments. Many important questions remain wide open. We present context-specific problems in the relevant sections, and also outline general interesting directions for future research in Section 9.

Before presenting our main results, we overview related work (Section 2) and provide a detailed description of our model (Section 3).

2 Related Work

Our work relates to many ideas in game theory and in distributed computing. We discuss game theoretic work on adaptive heuristics and on asynchrony, and also distributed computing work on fault tolerance and self stabilization. We also highlight the application areas we consider.

Adaptive heuristics. Much work in game theory and economics deals with adaptive heuristics (see Hart [16] and references therein). Generally speaking, this long line of research explores the "convergence" of simple and myopic rules of behavior (*e.g.*, best-response/fictitious-play/no-regret dynamics) to an "equilibrium". However, with few exceptions (see below), such analysis has so far primarily concentrated on synchronous environments in which steps take place simultaneously or in some other predetermined prescribed order. In contrast, we explore adaptive heuristics in asynchronous environments, which are more realistic for many applications.

Game-theoretic work on asynchronous environments. Some game-theoretic work on re-

peated games considers "asynchronous moves".[1] (see [23,34], among others, and references therein). Such work does not explore the behavior of dynamics, but has other research goals (*e.g.*, characterizing equilibria, establishing Folk theorems). We are, to the best of our knowledge, the first to study the effects of asynchrony (in the broad distributed computing sense) on the convergence of *game dynamics* to equilibria.

Fault-tolerant computation. We use ideas and techniques from work in distributed computing on protocol termination in asynchronous computational environments where nodes and communication channels are possibly faulty. Protocol termination in such environments, initially motivated by multi-processor computer architectures, has been extensively studied in the past three decades [2,4,7,12,20,29], as nicely surveyed in [11,24]. Fischer, Lynch and Paterson [12] showed, in a landmark paper, that a broad class of failure-resilient consensus protocols cannot provably terminate. Intuitively, the risk of protocol nontermination in [12] stems from the possibility of failures; a computational node cannot tell whether another node is silent due to a failure or is simply taking a long time to react. Our focus here is, in contrast, on failure-free environments.

Self stabilization. The concept of self stabilization is fundamental to distributed computing and dates back to Dijkstra, 1973 (see [8] and references therein). Convergence of adaptive heuristics to an "equilibrium" in our model can be viewed as the self stabilization of such dynamics (where the "equilibrium points" are the legitimate configurations). Our formulation draws ideas from work in distributed computing (*e.g.*, Burns' distributed daemon model) and in networking research [14] on self stabilization.

Applications. We discuss the implications of our non-convergence result across a wide variety of applications, that have previously been studied: convergence of game dynamics (see, *e.g.*, [18,19]); asynchronous circuits (see, *e.g.*, [6]); diffusion of innovations, behaviors, *etc.*, in social networks (see Morris [26] and also [21]); interdomain routing [14,30]; and congestion control [13].

3 The Model

We now present our model for analyzing adaptive heuristics in asynchronous environments.

Computational nodes interacting. There is an *interaction system* with n *computational nodes*, $1, \ldots, n$. Each computational node i has an *action space* A_i. Let $A = \times_{j \in [n]} A_j$, where $[n] = \{1, \ldots, n\}$. Let $A_{-i} = \times_{j \in [n] \setminus \{i\}} A_j$. Let $\Delta(A_i)$ be the set of all probability distributions over the actions in A_i.

Schedules. There is an infinite sequence of discrete time steps $t = 1, \ldots$. A *schedule* is a function σ that maps each $t \in \mathbb{N}_+ = \{1, 2, \ldots\}$ to a nonempty set of computational nodes: $\sigma(t) \subseteq [n]$. Informally, σ determines (when we consider the dynamics of the system) which nodes are *activated* in each time-step. We say that a schedule σ is *fair* if each node i is activated infinitely many times in σ, *i.e.*, $\forall i \in [n]$, there are infinitely many $t \in \mathbb{N}_+$ such that $i \in \sigma(t)$. For $r \in \mathbb{N}_+$, we say that a schedule σ is *r-fair* if each node is activated at least once in every sequence of r consecutive time steps, *i.e.*, if, for every $i \in [n]$ and $t_0 \in \mathbb{N}_+$, there is at least one value $t \in \{t_0, t_0 + 1, \ldots, t_0 + r - 1\}$ for which $i \in \sigma(t)$.

History and reaction functions. Let $H_0 = \emptyset$, and let $H_t = A^t$ for every $t \geq 1$. Intuitively, an

[1] Often, the term asynchrony merely indicates that players are not all activated at each time step, and thus is used to describe environments where only one player is activated at a time ("alternating moves"), or where there is a probability distribution that determines who is activated when.

element in H_t represents a possible *history* of interaction at time step t. For each node i, there is an infinite sequence of functions $f_i = (f_{(i,1)}, f_{(i,2)}, \ldots, f_{(i,t)}, \ldots)$ such that, for each $t \in \mathbb{N}_+$, $f_{(i,t)} : H_t \to \Delta(A_i)$; we call f_i the *reaction function* of node i. As discussed below, f_i captures i's way of responding to the history of interaction in each time step.

Restrictions on reaction functions. We now present five possible restrictions on reaction functions: determinism, self-independence, bounded recall, stationarity and historylessness.

1. **Determinism:** a reaction function f_i is *deterministic* if, for each input, f_i outputs a single action (that is, a probability distribution where a single action in A_i has probability 1).
2. **Self-independence:** a reaction function f_i is *self-independent* if node i's own (past and present) actions do not affect the outcome of f_i. That is, a reaction function f_i is self-independent if for every $t \geq 1$ there exists a function $g_t : A_{-i}^t \to \Delta(A_i)$ such that $f_{(i,t)} \equiv g_t$.
3. **k-recall and stationarity:** a node i has *k-recall* if its reaction function f_i only depends on the k most recent time steps, *i.e.*, for every $t \geq k$, there exists a function $g : H_k \to \Delta(A_i)$ such that $f_{(i,t)}(x) = g(x_{|k})$ for each input $x \in H_t$ ($x_{|k}$ here denotes the last k coordinates, *i.e.*, n-tuples of actions, of x). We say that a k-recall reaction function is *stationary* if the time counter t is of no importance. That is, a k-recall reaction function is stationary if there exists a function $g : H_k \to \Delta(A_i)$ such that *for all* $t \geq k$, $f_{(i,t)}(x) = g(x_{|k})$ for each input $x \in H_t$.
4. **Historylessness:** a reaction function f_i is *historyless* if f_i is 1-recall *and* stationary, that is, if f_i only depends on i's and on i's neighbors' most recent actions.

Dynamics. We now define *dynamics* in our model. Intuitively, there is some initial state (history of interaction) from which the interaction system evolves, and, in each time step, some subset of the nodes reacts to the past history of interaction. This is captured as follows. Let $s^{(0)}$, that shall be called the "*initial state*", be an element in H_w, for some positive $w \in \mathbb{N}$. Let σ be a schedule. We now describe the "$(s^{(0)}, \sigma)$-*dynamics*". The system's evolution starts at time $t = w + 1$, when each node $i \in \sigma(w + 1)$ simultaneously chooses an action according to $f_{(i,w+1)}$, *i.e.*, node i randomizes over the actions in A_i according to $f_{(i,w+1)}(s^{(0)})$. We now let $s^{(1)}$ be the element in H^{w+1} for which the first w coordinates (n-tuples of nodes' actions) are as in $s^{(0)}$ and the last coordinate is the n-tuple of *realized* nodes' actions at the end of time step $t = w + 1$. Similarly, in each time step $t > w+1$, each node in $\sigma(t)$ updates its action according to $f_{(i,t)}$, based on the past history $s^{(t-w-1)}$, and nodes' realized actions at time t, combined with $s^{(t-w-1)}$, define the history of interaction at the end of time step t, $s^{(t-w)}$.

Convergence and convergent systems. We say that nodes' actions *converge* under the $(s^{(0)}, \sigma)$-dynamics if there exist some positive $t_0 \in \mathbb{N}$, and some action profile $a = (a_1, \ldots, a_n)$, such that, for all $t > t_0$, $s^{(t)} = a$. The dynamics is then said to converge to a, and a is called a "*stable state*" (for the $(s^{(0)}, \sigma)$-dynamics), *i.e.*, intuitively, a stable state is a global action state that, once reached, remains unchanged. We say that the interaction system is *convergent* if, for all initial states $s^{(0)}$ and *fair* schedules σ, the $(s^{(0)}, \sigma)$-dynamics converges. We say that the system is *r-convergent* if, for all initial states $s^{(0)}$ and *r-fair* schedules σ, the $(s^{(0)}, \sigma)$-dynamics converges.

Update messages. Observe that, in our model, nodes' actions are *immediately observable* to other nodes at the end of each time step ("*perfect monitoring*"). While this is clearly unrealistic in some important real-life contexts (*e.g.*, some of the environments considered below), this restriction only strengthens our main results, that are impossibility results.

Deterministic historyless dynamics. Of special interest to us is the case that all reaction

functions are deterministic and historyless. We observe that, in this case, stable states have a simple characterization. Each reaction function f_i is deterministic and historyless and so can be specified by a function $g_i : A \to A_i$. Let $g = (g_1, \ldots, g_n)$. Observe that the set of all stable states (for all possible dynamics) is precisely the set of all fixed points of g. Below, when describing nodes' reaction functions that are deterministic and historyless we sometimes abuse notation and identify each f_i with g_i (treating f_i as a function from A to A_i). In addition, when all the reaction functions are also self-independent we occasionally treat each f_i as a function from A_{-i} to A_i.

4 Non-Convergence Result

We now present a general impossibility result for convergence of nodes' actions under bounded-recall dynamics in asynchronous, distributed computational environments.

Theorem 4.1. *If each reaction function has bounded recall and is self-independent then the existence of multiple stable states implies that the system is not convergent.*

We note that this result holds even if nodes' reaction functions are not stationary and are randomized (randomized initial states and activations are discussed in Section 6). We present the proof of Theorem 4.1 in Appendix F. We now discuss some aspects of our impossibility result.

Neither bounded recall nor self-independence alone implies non-convergence We show that the statement of Theorem 4.1 does not hold if either the bounded-recall restriction, or the self-independence restriction, is removed.

Example 4.2. **(the bounded-recall restriction cannot be removed)** There are two nodes, 1 and 2, each with the action space $\{x, y\}$. The deterministic and self-independent reaction functions of the nodes are as follows: node 2 always chooses node 1's action; node 1 will choose y if node 2's action changed from x to y in the past, and x otherwise. Observe that node 1's reaction function is not bounded-recall but can depend on the *entire* history of interaction. We make the observations that the system is safe and has two stable states. Observe that if node 1 chooses y at some point in time due to the fact that node 2's action changed from x to y, then it shall continue to do so thereafter; if, on the other hand, 1 never does so, then, from some point in time onwards, node 1's action is constantly x. In both cases, node 2 shall have the same action as node 1 eventually, and thus convergence to one of the two stable states, (x, x) and (y, y), is guaranteed. Hence, two stable states exist and the system is convergent nonetheless

Example 4.3. **(the self-independence restriction cannot be removed)** There are two nodes, 1 and 2, each with action set $\{x, y\}$. Each node i's a deterministic and historyless reaction function f_i is as follows: $f_i(x, x) = y$; in all other cases the node always (re)selects its current action (*e.g.*, $f_1(x, y) = x$, $f_2(x, y) = y$). Observe that the system has three stable states, namely all action profiles but (x, x), yet can easily be seen to be convergent.

Connections to consensus protocols. We now briefly discuss the interesting connections between Theorem 4.1 and the non-termination result for failure-resilient consensus protocols in [12]. We elaborate on this topic in Appendix A. Fischer *et al.* [12] explore when a group of processors can reach a consensus even in the presence of failures, and exhibit a breakthrough non-termination result. Our proof of Theorem 4.1 uses a valency argument—an idea introduced in the proof of the non-termination result in [12].

Intuitively, the risk of protocol non-termination in [12] stems from the possibility of failures; a computational node cannot tell whether another node is silent due to a failure or is simply taking a long time to react. We consider environments in which nodes/communication channels cannot fail, and so each node is guaranteed that all other nodes react after "sufficiently long" time. This guarantee makes reaching a consensus in the environment of [12] easily achievable (see Appendix A). Unlike the results in [12], the possibility of nonconvergence in our framework stems from limitations on nodes' behaviors. Hence, there is no immediate translation from the result in [12] to ours (and vice versa). To illustrate this point, we observe that in both Example 4.2 and Example 4.3, there exist two stable states and an initial state from which both stable states are reachable (a "bivalent state" [12]), yet the system is convergent (see Appendix A). This should be contrasted with the result in [12] that establishes that the existence of an initial state from which two distinct outcomes are reachable implies the existence of a non-terminating execution.

We investigate the link between consensus protocols and our framework further in Appendix F, where we take an axiomatic approach. We introduce a condition—"*Independence of Decisions*" (IoD)—that holds for both fault-resilient consensus protocols and for bounded-recall self-independent dynamics. We then factor the arguments in [12] through IoD to establish a non-termination result that holds for both contexts, thus unifying the treatment of these dynamic computational environments.

5 Games, Circuits, Networks, and Beyond

We present implications of our impossibility result in Section 4 for several well-studied environments: game theory, circuit design, social networks and Internet protocols. We now briefly summarize these implications, that, we believe, are themselves of independent interest. See Appendix B for a detailed exposition of the results in this section.

Game theory. Our result, when cast into game-theoretic terminology, shows that if players' choices of strategies are not synchronized, then the existence of two (or more) pure Nash equilibria implies that a broad class of game dynamics (*e.g.*, best-response dynamics with consistent tie-breaking) are not guaranteed to reach a pure Nash equilibrium. This result should be contrasted with positive results for such dynamics in the traditional synchronous game-theoretic environments.

Theorem 5.1. *If there are two (or more) pure Nash equilibria in a game, then all bounded-recall self-independent dynamics can oscillate indefinitely for asynchronous player activations.*

Corollary 5.2. *If there are two (or more) pure Nash equilibria in a game, then best-response dynamics, and bounded-recall best-response dynamics (studied in [35]), with consistent tie-breaking, can fail to converge to an equilibrium in asynchronous environments.*

Circuits. Work on asynchronous circuits in computer architectures research explores the implications of asynchrony for circuit design [6]. We observe that a logic gate can be regarded as executing an inherently historyless reaction function that is independent of the gate's past and present "state". Thus, we show that our result has implications for the stabilization of asynchronous circuits.

Theorem 5.3. *If two (or more) stable Boolean assignments exist for an asynchronous Boolean circuit, then that asynchronous circuit is not inherently stable.*

Social networks. Understanding the ways in which innovations, ideas, technologies, and practices, disseminate through social networks is fundamental to the social sciences. We consider the classic economic setting [26] (that has lately also been approached by computer scientists [21]) where each decision maker has two technologies $\{A, B\}$ to choose from, and each node in the social network wishes to have the same technology as the majority of his "friends" (neighboring nodes in the social network). We exhibit a general impossibility result for this environment.

Theorem 5.4. *In every social network, the diffusion of technologies can potentially never converge to a stable global state.*

Networking. We consider two basic networking environments: (1) routing with the Border Gateway Protocol (BGP), that is the "glue" that holds together the smaller networks that make up the Internet; and (2) the fundamental task of congestion control in communication networks, that is achieved through a combination of mechanisms on *end-hosts* (*e.g.*, TCP), and on *switches/routers* (*e.g.*, RED and WFQ). We exhibit non-termination results for both these environments.

We abstract a recent result in [30] and prove that this result extends to several BGP-based *multipath routing* protocols that have been proposed in the past few years.

Theorem 5.5. *[30] If there are multiple stable routing trees in a network, then BGP is not safe on that network.*

We consider the model for analyzing dynamics of congestion presented in [13]. We present the following result.

Theorem 5.6. *If there are multiple capacity-allocation equilibria in the network then dynamics of congestion can oscillate indefinitely.*

6 r-Convergence and Randomness

We now consider the implications for convergence of two natural restrictions on schedules: r-fairness and randomization. See Appendix C for a detailed exposition of the results in this section.

Snakes in boxes and r-convergence. Theorem 4.1 deals with convergence and not r-convergence, and thus does not impose restrictions on the number of consecutive time steps in which a node can be nonactive. What happens if there is an upper bound on this number, r? We now show that if $r < n - 1$ then sometimes convergence of historyless and self-independent dynamics is achievable even in the presence of multiple stable states (and so our impossibility result does not extend to this setting).

Example 6.1. **(a system that is convergent for $r < n - 1$ but nonconvergent for $r = n - 1$)** There are $n \geq 2$ nodes, $1, \ldots, n$, each with the action space $\{x, y\}$. Nodes' deterministic, historyless and self-independent reaction functions are as follows. $\forall i \in [n]$, $f_i(x^{n-1}) = x$ and f_i always outputs y otherwise. Observe that there exist two stable states: x^n and y^n. Observe that if $r = n - 1$ then the following oscillation is possible. Initially, only node 1's action is y and all other nodes' actions are x. Then, nodes 1 and 2 are activated and, consequently, node 1's action becomes x and node 2's action becomes y. Next, nodes 2 and 3 are activated, and thus 2's action becomes x and 3's action becomes y. Then $3, 4$ are activated, then $4, 5$, and so on (traversing all nodes over and over again in cyclic order). This goes on indefinitely, never reaching one of the two stable states. Observe

that, indeed, each node is activated at least once within every sequence of $n-1$ consecutive time steps. We observe however, that if $r < n-1$ then convergence is guaranteed. To see this, observe that if at some point in time there are at least two nodes whose action is y, then convergence to y^n is guaranteed. Clearly, if all nodes' action is x then convergence to x^n is guaranteed. Thus, an oscillation is possible only if, in each time step, *exactly* one node's action is y. Observe that, given our definition of nodes' reaction functions, this can only be if the activation sequence is (essentially) as described above, *i.e.*, exactly two nodes are activated at a time. Observe also that this kind of activation sequence is impossible for $r < n-1$.

What about $r > n$? We use classical results in combinatorics regarding the size of a "*snake-in-the-box*" in a hypercube [1] to construct systems are r-convergent for exponentially-large r's, but are not convergent in general.

Theorem 6.2. *Let $n \in \mathbb{N}$ be sufficiently large. There exists a system G with n nodes, in which each node i has two possible actions and each f_i is deterministic, historyless and self-independent, such that G is r-convergent for $r \in \Omega(2^n)$, but G is not $(r+1)$-convergent.*

We note that the construction in the proof of Theorem 6.2 is such that there is a unique stable state. We believe that the same ideas can be used to prove the same result for systems with multiple stable states but the exact way of doing this eludes us at the moment, and is left as an open question.

Problem 6.3. *Prove that for every sufficiently large $n \in \mathbb{N}$, there exists a system G with n nodes, in which each node i has two possible actions, each f_i is deterministic, historyless and self-independent, and G has multiple stable states, such that G is r-convergent for $r \in \Omega(2^n)$ but G is not $(r+1)$-convergent.*

Does random choice (of initial state and schedule) help? Theorem 4.1 tells us that, for a broad class of dynamics, a system with multiple stable states is nonconvergent if the initial state and the node-activation schedule are chosen adversarially. Can we guarantee convergence if the initial state and schedule are chosen *at random*?

Example 6.4. **(random choice of initial state and schedule might not help)** There are n nodes, $1, \ldots, n$, and each node has action space $\{x, y, z\}$. The (deterministic, historyless and self-independent) reaction function of each node $i \in \{3, \ldots, n\}$ is such that $f_i(x^{n-1}) = x$; $f_i(z^{n-1}) = z$; and $f_i = y$ for all other inputs. The (deterministic, historyless and self-independent) reaction function of each node $i \in \{1, 2\}$ is such that $f_i(x^{n-1}) = x$; $f_i(z^{n-1}) = z$; $f_i(xy^{n-2}) = y$; $f_i(y^{n-1}) = x$; and $f_i = y$ for all other inputs. Observe that there are exactly two stable states: x^n and z^n. Observe also that if nodes' actions in the initial state do not contain at least $n-1$ x's, or at least $n-1$ z's, then, from that moment forth, each activated node in the set $\{3, \ldots, n\}$ will choose the action y. Thus, eventually the actions of all nodes in $\{3, \ldots, n\}$ shall be y, and so none of the two stable states will be reached. Hence, there are 3^n possible initial states, such that only from $4n+2$ can a stable state be reached. When choosing the initial state uniformly at random the probability of landing on a "good" initial state (in terms of convergence) is thus exponentially small.

7 Complexity of Asynchronous Dynamics

We now explore the communication complexity and computational complexity of determining whether a system is convergent. We present hardness results in both models of computation even

for the case of deterministic and historyless adaptive heuristics. See Appendix D for a detailed exposition of the results in this section.

We first present the following communication complexity result whose proof relies on combinatorial "snake-in-the-box" constructions [1].

Theorem 7.1. *Determining if a system with n nodes, each with 2 actions, is convergent requires $\Omega(2^n)$ bits. This holds even if all nodes have deterministic, historyless and self-independent reaction functions.*

The above communication complexity hardness result required the representation of the reaction functions to (potentially) be exponentially long. What if the reaction functions can be succinctly described? We now present a strong computational complexity hardness result for the case that each reaction function f_i is deterministic and historyless, and is given explicitly in the form of a boolean circuit (for each $a \in A$ the circuit outputs $f_i(a)$). We prove the following result.

Theorem 7.2. *Determining if a system with n nodes, each with a deterministic and historyless reaction function, is convergent is PSPACE-complete.*

Our computational complexity result shows that even if nodes' reaction functions can be succinctly represented, determining whether the system is convergent is PSPACE-complete. This result, alongside its computational implications, implies that we cannot hope to have short "witnesses" of guaranteed asynchronous convergence (unless PSPACE $\subseteq$ NP). Proving the above PSPACE-completeness result for the case self-independent reaction functions seems challenging.

Problem 7.3. *Prove that determining if a system with n nodes, each with a deterministic self-independent and historyless reaction function, is convergent is PSPACE-complete.*

8 Some Basic Observations Regarding No-Regret Dynamics

Regret minimization is fundamental to learning theory, and has strong connections to game-theoretic solution concepts; if each player in a repeated game executes a no-regret algorithm when selecting strategies, then convergence to an equilibrium is guaranteed in a variety of interesting contexts. The meaning of convergence, and the type of equilibrium reached, vary, and are dependent on the restrictions imposed on the game and on the notion of regret. Work on no-regret dynamics traditionally considers environments where all nodes are "activated" at each time step. We make the simple observation that, switching our attention to r-fair schedules (for every $r \in N_+$), if an algorithm has no regret in the classic setting, then it has no regret in this new setting as well (for all notions of regret). Hence, positive results from the regret-minimization literature extend to this asynchronous environment. See [3] for a thorough explanation about no-regret dynamics and see Appendix E for a detailed explanation about our observations. We now mention two implications of our observation and highlight two open problems regarding regret minimization.

Observation 8.1. *When all players in a zero-sum game use no-external-regret algorithms then approaching or exceeding the minimax value of the game is guaranteed.*

Observation 8.2. *When all players in a (general) game use no-swap-regret algorithms the empirical distribution of joint players' actions converges to a correlated equilibrium of the game.*

Problem 8.3. *Give examples of repeated games for which there exists a schedule of player activations that is not r-fair for any $r \in N_+$ for which regret-minimizing dynamics do not converge to an equilibrium (for different notions of regret/convergence/equilibria).*

Problem 8.4. *When is convergence of no-regret dynamics to an equilibrium guaranteed (for different notions of regret/convergence/equilibria) for all r-fair schedules for non-fixed r's, that is, if when r is a function of t?*

9 Future Research

In this paper, we have taken the first steps towards a complete understanding of distributed computing with adaptive heuristics. We proved a general non-convergence result and several hardness results within this model, and also discussed some important aspects such as the implications of fairness and randomness, as well as applications to a variety of settings. We believe that we have but scratched the surface in the exploration of the convergence properties of simple dynamics in asynchronous computational environments, and many important questions remain wide open. We now outline several interesting directions for future research.

Other heuristics, convergence notions, equilibria. We have considered specific adaptive heuristics, notions of convergence, and kinds of equilibria. Understanding the effects of asynchrony on other adaptive heuristics (*e.g.*, better-response dynamics, fictitious play), for other notions of convergence (*e.g.*, of the empirical distributions of play), and for other kinds of equilibria (*e.g.*, mixed Nash equilibria, correlated equilibria) is a broad and challenging direction for future research.

Outdated and private information. We have not explicitly considered the effects of making decisions based on outdated information. We have also not dealt with the case that nodes' behaviors are dependent on private information, that is, the case that the dynamics are "uncoupled" [18,19].

Other notions of asynchrony. We believe that better understanding the role of degrees of fairness, randomness, and other restrictions on schedules from distributed computing literature, in achieving convergence to equilibrium points is an interesting and important research direction.

Characterizing asynchronous convergence. We still lack characterizations of asynchronous convergence even for simple dynamics (*e.g.*, deterministic and historyless).[2]

Topological and knowledge-based approaches. Topological [4,20,29] and knowledge-based [15] approaches have been very successful in addressing fundamental questions in distributed computing. Can these approaches shed new light on the implications of asynchrony for adaptive heuristics?

Further exploring the environments in Section 5. We have applied our non-convergence result to the environments described in Section 5. These environments are of independent interest and are indeed the subject of extensive research. Hence, the further exploration of dynamics in these settings is important.

Acknowledgements

We thank Danny Dolev, Alex Fabrikant, Idit Keidar, Jonathan Laserson, Nati Linial, Yishay Mansour and Yoram Moses for helpful discussions. This work was initiated partly as a result of the DIMACS Special Focus on Communication Security and Information Privacy.

[2] Our PSPACE-completeness result in Section 7 eliminates the possibility of short witnesses of guaranteed asynchronous convergence unless PSPACE $\subseteq$ NP, but elegant characterizations are still possible.

References

[1] H. L. Abbott and M. Katchalski. On the construction of snake in the box codes. *Utilitas Mathematica*, 40:97-116, 1991.

[2] M. Ben-Or. Randomized agreement protocols. In *Fault-Tolerant Distributed Computing*, pages 72–83, 1986.

[3] A. Blum and Y. Mansour. Learning, regret minimization, and equilibria. In *Algorithmic Game Theory, Cambridge University Press*, 2007.

[4] E. Borowsky and E. Gafni. Generalized FLP impossibility result for t-resilient asynchronous computations. In *STOC '93: Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 91–100, 1993.

[5] K. M. Chandy and J. Misra. How processes learn. In *PODC '85: Proceedings of the fourth annual ACM symposium on Principles of distributed computing*, pages 204–214, 1985.

[6] A. Davis and S. M. Nowick. An introduction to asynchronous circuit design. Technical report, The Encyclopedia of Computer Science and Technology, 1997.

[7] D. Dolev, C. Dwork, and L. Stockmeyer. On the minimal synchronism needed for distributed consensus. *J. ACM*, 34(1):77–97, 1987.

[8] S. Dolev. Self stabilization. *MIT Press*, 2000.

[9] A. Fabrikant and C. H. Papadimitriou. The complexity of game dynamics: BGP oscillations, sink equilibria, and beyond. In *SODA '08: Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 844–853, 2008.

[10] J. Feigenbaum and S. Shenker. Distributed algorithmic mechanism design: recent results and future directions. In *DIALM '02: Proceedings of the 6th international workshop on Discrete algorithms and methods for mobile computing and communications*, pages 1–13, 2002.

[11] F. Fich and E. Ruppert. Hundreds of impossibility results for distributed computing. *Distributed Computing*, 16(2–3):121–163, 2003.

[12] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, 1985.

[13] P. B. Godfrey, M. Schapira, A. Zohar, and S. Shenker. Incentive compatibility and dynamics of congestion control. In *SIGMETRICS '10*, pages 95–106, 2010.

[14] T. G. Griffin, F. B. Shepherd, and G. Wilfong. The stable paths problem and interdomain routing. *IEEE/ACM Trans. Netw.*, 10(2):232–243, 2002.

[15] J. Y. Halpern and Y. Moses. Knowledge and common knowledge in a distributed environment. *JACM*, 37(3):549–587, 1990.

[16] S. Hart. Adaptive heuristics. *Econometrica*, 73:1401–1430, 2005.

[17] S. Hart and Y. Mansour. The communication complexity of uncoupled nash equilibrium procedures. In *STOC '07: Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 345–353, 2007.

[18] S. Hart and A. Mas-Colell. Uncoupled dynamics do not lead to Nash equilibrium. *American Economic Review*, 93(5):1830–1836, 2003.

[19] S. Hart and A. Mas-Colell. Stochastic uncoupled dynamics and Nash equilibrium. *Games and Economic Behavior*, 57(2):286–303, 2006.

[20] M. Herlihy and N. Shavit. The topological structure of asynchronous computability. *J. ACM*, 46(6):858–923, 1999.

[21] N. Immorlica, J. Kleinberg, M. Mahdian, and T. Wexler. The role of compatibility in the diffusion of technologies through social networks. In *EC '07: Proceedings of the 8th ACM conference on Electronic commerce*, pages 75–83, 2007.

[22] N. Kushman and S. Kandula and D. Katabi and B. Maggs. R-BGP: staying connected in a connected world In *NSDI '07: 4th USENIX Symposium on Networked Systems Design and Implementation*, 2007.

[23] R. Lagunoff and A. Matsui. Asynchronous choice in repeated coordination games. *Econometrica*, 65(6):1467-1478, 1997.

[24] N. Lynch. A hundred impossibility proofs for distributed computing. In *Proceedings of the Eighth Annual ACM Symposium on Principles of Distributed Computing*, pages 1–28, 1989.

[25] D. Monderer and L. S. Shapley. Potential games. *Games and Economic Behavior*, 14:124–143, 1996.

[26] S. Morris. Contagion. *Review of Economic Studies*, 67:5778, 2000.

[27] N. Nisan, M. Schapira, and A. Zohar. Asynchronous best-reply dynamics. In *WINE '08: Proceedings of the Workshop on Internet Economics*, pages 531–538, 2008.

[28] R. W. Rosenthal. A class of games possessing pure-strategy Nash equilibria. *Int. J. Game Theory*, 2:65-67, 1973.

[29] M. E. Saks and F. Zaharoglou. Wait-free k-set agreement is impossible: The topology of public knowledge. *SIAM J. Comput.*, 29(5):1449–1483, 2000.

[30] R. Sami, M. Schapira, and A. Zohar. Searching for stability in interdomain routing. In *INFOCOM*, 2009.

[31] G. Taubenfeld. On the nonexistence of resilient consensus protocols. *Inf. Process. Lett.*, 37(5):285–289, 1991.

[32] K. Varadhan, R. Govindan, and D. Estrin. Persistent route oscillations in inter-domain routing. *Computer Networks*, 32(1):1–16, 2000.

[33] Y. Wang, M. Schapira, and J. Rexford. Neighbor-specific BGP: more flexible routing policies while improving global stability. In SIGMETRICS '09: Proceedings of the eleventh international joint conference on Measurement and modeling of computer systems, pages 217–228, 2009.

[34] K. Yoon. The effective minimax value of asynchronously repeated games. *International Journal of Game Theory*, 32(4):431-442, 2004.

[35] A. Zapechelnyuk. Better-reply dynamics with bounded recall. *Mathematics of Operations Research*, 33:869–879, 2008.

A Connections to Consensus Protocols

There are interesting connections between our result and that of Fischer *et al.* [12] for *fault-resilient consensus protocols*. [12] studies the following environment: There is a group of *processes*, each with an initial value in $\{0,1\}$, that communicate with each other via *messages*. The objective is for all *non-faulty* processes to eventually agree on some value $x \in \{0,1\}$, where the "consensus" x must match the initial value of some process. [12] establishes that no consensus protocol is resilient to even a single failure. One crucial ingredient for the proof of the result in [12] is showing that there exists some initial configuration of processes' initial values such that, from that configuration, the resulting consensus can be both 0 and 1 (the outcome depends on the specific "schedule" realized). Our proof of Theorem 4.1 uses a valency argument—an idea introduced in the proof of the breakthrough non-termination result in [12] for consensus protocols.

Intuitively, the risk of protocol nontermination in [12] stems from the possibility of failures; a computational node cannot tell whether another node is silent due to a failure or is simply taking a long time to react. We consider environments in which nodes/communication channels do not fail. Thus, each node is guaranteed that after "sufficiently many" time steps all other nodes will react. Observe that in such an environment reaching a consensus is easy; one pre-specified node i (the "dictator") waits until it learns all other nodes' inputs (this is guaranteed to happen as failures are impossible) and then selects a value v_i and informs all other nodes; then, all other nodes select v_i. Unlike the results in [12], the possibility of nonconvergence in our framework stems from limitations on nodes' behaviors. We investigate the link between consensus protocols and our framework further in Appendix. F, where we take an axiomatic approach. We introduce a condition—"*Independence of Decisions*" (IoD)—that holds for both fault-resilient consensus protocols and for bounded-recall self-independent dynamics. We then factor the arguments in [12] through IoD to establish a non-termination result that holds for both contexts, thus unifying the treatment of these dynamic computational environments.

Hence, there is *no* immediate translation from the result in [12] to ours (and vice versa). To illustrate this point, let us revisit Example 4.2, in which the system is convergent, yet two stable states exist. We observe that in the example there is indeed an initial state from which both stable states are reachable (a "bivalent state" [12]). Consider the initial state (y,x). Observe that if node 1 is activated first (and alone), then it shall choose action x. Once node 2 is activated it shall then also choose x, and the resulting stable state shall be (x,x). However, if node 2 is activated first (alone), then it shall choose action y. Once 1 is activated it shall also choose action y, and the resulting stable state shall be (y,y). Observe that in Example 4.3 too there exists an action profile (x,x) from which multiple stable states are reachable yet the system is convergent.

B Games, Circuits, Networks, and Beyond

We present implications of our impossibility result in Section 4 for several well-studied environments: game theory, circuit design, social networks and Internet protocols.

B.1 Game Dynamics

The setting. There are n *players*, $1, \ldots, n$. Each player i has a *strategy set* S_i. Let $S = \times_{j \in N} S_j$, and let $S_{-i} = \times_{j \in [n] \setminus \{i\}} S_j$. Each player i has a *utility function* $u_i : S \to S_i$. For each $s_i \in S_i$ and $s_{-i} \in S_{-i}$ let (s_i, s_{-i}) denote the strategy profile in which player i's strategy is s_i and all other players' strategies are as in s_{-i}. Informally, a *pure Nash equilibrium* is a strategy profile from which no player wishes to unilaterally deviate.

Definition B.1. (pure Nash equilibria) We say that a strategy profile $\overline{s} = (\overline{s_1}, \ldots, \overline{s_n}) \in S$ is a *pure Nash equilibrium* if, for each player i, $\overline{s_i} \in argmax_{s_i \in S_i} u_i(s_i, \overline{s_{-i}})$.

One natural procedure for reaching a pure Nash equilibrium of a game is *best-response dynamics*: the process starts at some arbitrary strategy profile, and players take turns "best replying" to other players' strategies until no player wishes to change his strategy. Convergence of best-response dynamics to pure Nash equilibria is the subject of extensive research in game theory and economics, and both positive [25, 28] and negative [18, 19] results are known.

Traditionally, work in game theory on game dynamics (*e.g.*, best-response dynamics) relies on the explicit or implicit premise that players' actions are somehow synchronized (in some contexts play is sequential, while in others it is simultaneous). We consider the realistic scenario that there is no computational center than can synchronize players' selection of strategies. We cast the above setting into the terminology of Section 3 and exhibit an impossibility result for best-response, and more general, dynamics.

Computational nodes, action spaces. The *computational nodes* are the n players. The *action space* of each player i is his strategy set S_i.

Reaction functions, dynamics. Under best-response dynamics, each player constantly chooses a "best response" to the other players' most recent actions. Consider the case that players have consistent tie-breaking rules, *i.e.*, the best response is always unique, and depends only on the others' strategies. Observe that, in this case, players' behaviors can be formulated as deterministic, historyless, and self-independent reaction functions. The dynamic interaction between players is as in Section 3.

Existence of multiple pure Nash equilibria implies non-convergence of best-response dynamics in asynchronous environments. Theorem 4.1 implies the following result:

Theorem B.2. *If there are two (or more) pure Nash equilibria in a game, then asynchronous best-response dynamics can potentially oscillate indefinitely.*

In fact, Theorem 4.1 implies that the above non-convergence result holds even for the broader class of randomized, bounded-recall and self-independent game dynamics, and thus also to game dynamics such as best-response with bounded recall and consistent tie-breaking rules (studied in [35]).

B.2 Asynchronous Circuits

The setting. There is a Boolean circuit, represented as a directed graph G, in which vertices represent the circuit's inputs and the logic gates, and edges represent connections between the circuit's inputs and the logic gates and between logic gates. The activation of the logic gates is asynchronous. That is, the gates' outputs are initialized in some arbitrary way, and then the update of each gate's output, given its inputs, is uncoordinated and unsynchronized. We prove an impossibility result for this setting, which has been extensively studied (see [6]).

Computational nodes, action spaces. The computational nodes are the inputs and the logic gates. The *action space* of each node is $\{0, 1\}$.

Reaction functions, dynamics. Observe that each logic gate can be regarded as a function that only depends on its inputs' values. Hence, each logic gate can be modeled via a *reaction function*. Interaction between the different circuit components is as in Section 3.

Too much stability in circuits can lead to instability. *Stable states* in this framework are assignments of Boolean values to the circuit inputs and the logic gates that are consistent with each gate's truth table (reaction function). We say that a Boolean circuit is *inherently stable* if it is guaranteed to converge to a stable state regardless of the initial boolean assignment. The following theorem is derived from Theorem 4.1:

Theorem B.3. *If two (or more) stable Boolean assignments exist for an asynchronous Boolean circuit, then that asynchronous circuit is not inherently stable.*

B.3 Diffusion of Technologies in Social Networks

The setting. There is a social network of users, represented by a directed graph in which users are the vertices and edges correspond to friendship relationships. There are two competing technologies, X and Y. A user's utility from each technology depends on the number of that user's friends that use that technology; the more friends use that technology the more desirable that technology is to the user. That is, a user would always select the technology used by the majority of his friends. We are interested in the dynamics of the diffusion of technologies. Observe that if, initially, all users are using X, or all users are using Y, no user has an incentive to switch to a different technology. Hence, there are always (at least) two distinct "stable states" (regardless of the topology of the social network). Therefore, the terminology of Section 3 can be applied to this setting.

Computational nodes, actions spaces. The users are the *computational nodes*. Each user i's *action space* consists of the two technologies $\{X, Y\}$.

Reaction functions, dynamics. The reaction function of each user i is defined as follows: If at least half of i's friends use technology X, i selects technology X; otherwise, i selects technology Y. In our model of diffusion of technologies, users' choices of technology can be made simultaneously, as described in Section 3.

Instability of social networks. Theorem 4.1 implies the following:

Theorem B.4. *In every social network, the diffusion of technologies can potentially never converge to a stable global state.*

B.4 Interdomain Routing

The setting. The Internet is made up of smaller networks called *Autonomous Systems* (ASes). *Interdomain routing* is the task of establishing routes between ASes, and is handled by the *Border Gateway Protocol* (BGP). In the standard model for analyzing BGP dynamics [14], there is a network of *source* ASes that wish to send traffic to a unique *destination* AS d. Each AS i has a *ranking function* $<_i$ that specifies i's strict preferences over all simple (loop-free) routes leading from i to d.[3] Under BGP, each AS constantly selects the "best" route that is available to it. See [14] for more details. Guaranteeing *BGP safety*, *i.e.*, BGP convergence to a "stable" routing outcome is a fundamental desideratum that has been the subject of extensive work in both the networking and the standards communities. We now cast interdomain routing into the terminology of Section 3. We then obtain non-termination results for BGP and for proposals for new interdomain routing protocols (as corollaries of Theorem 4.1).

Computational nodes, action spaces. The ASes are the *computational nodes*. The *action space* of each node i, A_i, is the set of all simple (loop-free) routes between i and the destination d that are exportable to i, and the empty route $\emptyset$.

Reaction functions, dynamics. The *reaction function* f_i of node i outputs, for every vector α containing routes to d of all of i's neighbors, a route $(i,j)R_j$ such that (1) j is i's neighbor; (2) R_j is j's route in α; and (3) $R_j >_i R$ for all other routes R in α. If there is no such route R_j in α then f_i outputs $\emptyset$. Observe that the reaction function f_i is deterministic, self-independent and historyless. The interaction between nodes is as described in Section 3.

The multitude of stable routing trees implies global network instability. Theorem 4.1 implies a recent result of Sami *et al.* [30], that shows that the existence of two (or more) stable routing trees to which BGP can (potentially) converge implies that BGP is not safe. Importantly, the asynchronous model of Section 3 is significantly *more restrictive* than that of [30]. Hence, Theorem 4.1 implies the non-termination result of Sami *et al.*

Theorem B.5. *[30] If there are multiple stable routing trees in a network, then BGP is not safe on that network.*

Over the past few years, there have been several proposals for BGP-based *multipath routing* protocols, *i.e.*, protocols that enable each node (AS) to send traffic along multiple routes, *e.g.*, R-BGP [22] and Neighbor-Specific BGP [33] (NS-BGP). Under both R-BGP and NS-BGP each computational node's actions are independent of its own past actions and are based on bounded recall of past interaction. Thus, Theorem 4.1 implies the following:

Theorem B.6. *If there are multiple stable routing configurations in a network, then R-BGP is not safe on that network.*

Theorem B.7. *If there are multiple stable routing configurations in a network, then NS-BGP is not safe on that network.*

B.5 Congestion Control

The setting. We now present the model of congestion control, studied in [13]. There is a network of routers, represented by a directed graph $G = (V, E)$, where $|E| \geq 2$, in which vertices represent

[3] ASes rankings of routes also reflect each AS's *export policy* that specifies which routes that AS is willing to make available to each neighboring AS.

routers, and edges represent communication links. Each edge has capacity c_e. There are n *source-target pairs* of vertices (s_i, t_i), termed "*connections*", that represent communicating pairs of end-hosts. Each source-target pair (s_i, t_i) is connected via some *fixed* route, R_i. Each source s_i transmits at a *constant* rate $\gamma_i > 0$.[4] Routers have *queue management*, or *queueing*, policies, that dictate how traffic traversing a router's outgoing edge should be divided between the connections whose routes traverse that edge. The network is asynchronous and so routers' queueing decisions can be made simultaneously. See [13] for more details.

Computational nodes, action spaces The *computational nodes* are the edges. The *action space* of each edge e intuitively consists of all possible way to divide traffic going through e between the connections whose routes traverse e. More formally, for every edge e, let $N(e)$ be the number connections whose paths go through e. e's action space is then $A_i = \{x = (x_1, \ldots, x_{N(e)}) | x_i \in \mathbb{R}_{\geq 0}^{N(e)}$ and $\Sigma_i x_i \leq c_e\}$.

Reaction functions, dynamics. Each edge e's *reaction function*, f_e, models the queueing policy according to which e's capacity is shared: for every $N(e)$-tuple of nonnegative incoming flows $(w_1, w_2, \ldots, w_{N(e)})$, f_e outputs an action $(x_1, \ldots, x_{N(e)}) \in A_i$ such that $\forall i \in [N(e)]$ $w_i \geq x_i$ (a connection's flow leaving the edge cannot be bigger than that connection's flow entering the edge). The interaction between the edges is as described in Section 3.

Multiple equilibria imply potential fluctuations of connections' throughputs. [13] shows that, while one might expect that if sources transmit flow at a constant rate, flow will also be *received* at a constant rate, this is not necessarily the case. Indeed, [13] presents examples in which connections' throughputs can potentially fluctuate ad infinitum. Equilibria (which correspond to stable states in Section 3), are global configurations of connections' flows on edges such that connections' incoming and outgoing flows on each edge are consistent with the queue management policy of the router controlling that edge. Using Theorem 4.1, we can obtain the following impossibility result:

Theorem B.8. *If there are multiple capacity-allocation equilibria in the network then dynamics of congestion can potentially oscillate indefinitely.*

C r-Convergence and Randomness

We now consider the implications for convergence of two natural restrictions on schedules: r-fairness and randomization.

C.1 Snakes in Boxes and r-Convergence.

Theorem 4.1 deals with convergence and not r-convergence, and thus does not impose restrictions on the number of consecutive time steps in which a node can be nonactive. What happens if there is an upper bound on this number, r? We now show that if $r < n - 1$ then sometimes convergence of historyless and self-independent dynamics is achievable even in the presence of multiple stable states (and so our impossibility result breaks).

[4] This is modeled via the addition of an edge $e = (u, s_i)$ to G, such that $c_e = \gamma_i$, and u has no incoming edges.

Example C.1. **(a system that is convergent for $r < n-1$ but nonconvergent for $r = n-1$)** There are $n \geq 2$ nodes, $1, \ldots, n$, each with the action space $\{x, y\}$. Nodes' deterministic, historyless and self-independent reaction functions are as follows. $\forall i \in [n]$, $f_i(x^{n-1}) = x$ and f_i always outputs y otherwise. Observe that there exist two stable states: x^n and y^n. Observe that if $r = n-1$ then the following oscillation is possible. Initially, only node 1's action is y and all other nodes' actions are x. Then, nodes 1 and 2 are activated and, consequently, node 1's action becomes x and node 2's action becomes y. Next, nodes 2 and 3 are activated, and thus 2's action becomes x and 3's action becomes y. Then $3, 4$ are activated, then $4, 5$, and so on (traversing all nodes over and over again in cyclic order). This goes on indefinitely, never reaching one of the two stable states. Observe that, indeed, each node is activated at least once within every sequence of $n-1$ consecutive time steps. We observe however, that if $r < n-1$ then convergence is guaranteed. To see this, observe that if at some point in time there are at least two nodes whose action is y, then convergence to y^n is guaranteed. Clearly, if all nodes' action is x then convergence to x^n is guaranteed. Thus, an oscillation is possible only if, in each time step, *exactly* one node's action is y. Observe that, given our definition of nodes' reaction functions, this can only be if the activation sequence is (essentially) as described above, *i.e.*, exactly two nodes are activated at a time. Observe also that this kind of activation sequence is impossible for $r < n-1$.

What about $r > n$? We use classical results in combinatorics regarding the size of a "*snake-in-the-box*" in a hypercube [1] to show that some systems are r-convergent for exponentially-large r's, but are not convergent in general.

Theorem 6.2 1. Let $n \in \mathbb{N}$ be sufficiently large. There exists a system G with n nodes, in which each node i has two possible actions and each f_i is deterministic, historyless and self-independent, such that

1. G is r-convergent for $r \in \Omega(2^n)$;
2. G is not $(r+1)$-convergent.

Proof. Let the action space of each of the n nodes be $\{x, y\}$. Consider the possible action profiles of nodes $3, \ldots, n$, *i.e.*, the set $\{x, y\}^{n-2}$. Observe that this set of actions can be regarded as the $(n-2)$-hypercube Q_{n-2}, and thus can be visualized as the graph whose vertices are indexed by the binary $(n-2)$-tuples and such that two vertices are adjacent iff the corresponding $(n-2)$-tuples differ in exactly one coordinate.

Definition C.2. (chordless paths, snakes) A *chordless path* in a hypercube Q_n is a path $P = (v_0, \ldots, v_w)$ such that for each v_i, v_j on P, if v_i and v_j are neighbors in Q_n then $v_j \in \{v_{i-1}, v_{i+1}\}$. A *snake* in a hypercube is a simple chordless cycle.

The following result is due to Abbot and Katchalski [1].

Theorem C.3. *[1] Let $t \in \mathbb{N}$ be sufficiently large. Then, the size $|S|$ of a maximal snake in the z-hypercube Q_z is at least $\lambda \times 2^z$ for some $\lambda \geq 0.3$.*

Hence, the size of a maximal snake in the Q_{n-2} hypercube is $\Omega(2^n)$. Let S be a maximal snake in $\{x, y\}^{n-2}$. W.l.o.g we can assume that x^{n-2} is on S (otherwise we can rename nodes' actions so as to achieve this). Nodes deterministic, historyless and self-independent are as follows:

- Node $i \in \{1, 2\}$: $f_i(x^{n-1}) = x$; $f_i = y$ otherwise.

- Node $i \in \{3, \ldots, n\}$: if the actions of nodes 1 and 2 are both y then the action y is chosen, *i.e.*, $f_i(yy * \ldots *) = y$; otherwise, f_i only depends on the actions of nodes in $\{3, \ldots, n\}$ and therefore to describe f_i it suffices to orient the edges of the hypercube Q_{n-2} (an edge from one vertex to another vertex that differs from it in the ith coordinate determines the outcome of f_i for both). This is done as follows: orient the edges in S so as to create a cycle (in one of two possible ways); orient edges between vertices not in S to vertices in S towards the vertices in S; orient all other edges arbitrarily.

Observation C.4. *x^n is the unique stable state of the system.*

Observation C.5. *If, at some point in time, both nodes 1 and 2's actions are y then convergence to the y^n stable state is guaranteed.*

Claim C.6. *If there is an oscillation then there must be infinitely many time steps in which the actions of nodes $2, \ldots, n$ are x^{n-1}.*

Proof. Consider the case that the statement does not hold. In that case, from some moment forth, node 1 never sees the actions x^{n-1} and so will constantly select the action y. Once that happens, node 2 shall also not see the actions x^{n-1} and will thereafter also select y. Convergence to y^n is then guaranteed. □

We now show that the system is convergent for $r < |S|$, but is nonconvergent if $r = |S|$. The theorem follows.

Claim C.7. *If $r < |S|$ then convergence to the stable state y^n is guaranteed.*

Proof. Observation C.6 establishes that in an oscillation there must be infinitely many time steps in which the actions of nodes $2, \ldots, n$ are x^{n-1}. Consider one such moment in time. Observe that in the subsequent time steps nodes' action profiles will inevitably change as in S (given our definition of nodes' $3, \ldots, n$ reaction functions). Thus, once the action profile is no longer x^{n-1} there are at least $|S| - 1$ time steps until it goes back to being x^{n-1}. Observe that if 1 and 2 are activated at some point in the intermediate time steps (which is guaranteed as $r < |S|$) then the actions of both shall be y and so convergence to y^n is guaranteed. □

Claim C.8. *If $r = |S|$ then an oscillation is possible.*

Proof. The oscillation is as follows. Start at x^n and activate both 1 and 2 (this will not change the action profile). In the $|S| - 1$ subsequent time steps activate all nodes but 1 and 2 until x^n is reached again. Repeat ad infinitum. □

□

We note that the construction in the proof of Theorem 6.2 is such that there is a unique stable state. We believe that the same ideas can be used to prove the same result for systems with multiple stable states but the exact way of doing this eludes us at the moment, and is left as an open question.

Problem C.9. *Prove that for every sufficiently large $n \in \mathbb{N}$, there exists a system G with n nodes, in which each node i has two possible actions and each f_i is deterministic, historyless and self-independent, such that*

1. *G is r-convergent for $r \in \Omega(2^n)$;*
2. *G is not $(r+1)$-convergent;*
3. *There are multiple stable states in G.*

C.2 Does Random Choice (of Initial State and Schedule) Help?

Theorem 4.1 tells us that a system with multiple stable states is nonconvergent if the initial state and the node-activation schedule are chosen adversarially. Can we guarantee convergence if the initial state and schedule are chosen *at random*?

Example C.10. **(random choice of initial state and schedule might not help)** There are n nodes, $1, \ldots, n$, and each node has action space $\{x, y, z\}$. The (deterministic, historyless and self-independent) reaction function of each node $i \in \{3, \ldots, n\}$ is such that $f_i(x^{n-1}) = x$; $f_i(z^{n-1}) = z$; and $f_i = y$ for all other inputs. The (deterministic, historyless and self-independent) reaction function of each node $i \in \{1, 2\}$ is such that $f_i(x^{n-1}) = x$; $f_i(z^{n-1}) = z$; $f_i(xy^{n-2}) = y$; $f_i(y^{n-1}) = x$; and $f_i = y$ for all other inputs. Observe that there are exactly two stable states: x^n and z^n. Observe also that if nodes' actions in the initial state do not contain at least $n-1$ x's, or at least $n-1$ z's, then, from that moment forth, each activated node in the set $\{3, \ldots, n\}$ will choose the action y. Thus, eventually the actions of all nodes in $\{3, \ldots, n\}$ shall be y, and so none of the two stable states will be reached. Hence, there are 3^n possible initial states, such that only from $4n+2$ can a stable state be reached.

Example C.10 presents a system with multiple stable states such that from most initial states *all* possible choices of schedules do not result in a stable state. Hence, when choosing the initial state uniformly at random the probability of landing on a "good" initial state (in terms of convergence) is exponentially small.

D Complexity of Asynchronous Dynamics

We now explore the communication complexity and computational complexity of determining whether a system is convergent. We present hardness results in both models of computation even for the case of deterministic and historyless adaptive heuristics. Our computational complexity result shows that even if nodes' reaction functions can be succinctly represented, determining whether the system is convergent is PSPACE-complete. This intractability result, alongside its computational implications, implies that we cannot hope to have short "witnesses" of guaranteed asynchronous convergence (unless PSPACE $\subseteq$ NP).

D.1 Communication Complexity

We prove the following communication complexity result, that shows that, in general, determining whether a system is convergent cannot be done efficiently.

Theorem D.1. *Determining if a system with n nodes, each with 2 actions, is convergent requires $\Omega(2^n)$ bits. This holds even if all nodes have deterministic, historyless and self-independent reaction functions.*

Proof. To prove our result we present a reduction from the following well-known problem in communication complexity theory.

2-party SET DISJOINTNESS: There are two parties, Alice and Bob. Each party holds a subset of $\{1, \ldots, q\}$; Alice holds the subset E^A and Bob holds the subset E^B. The objective is to determine whether $E^A \cap E^B = \emptyset$. The following is well known.

Theorem D.2. *Determining whether $E^A \cap E^B = \emptyset$ requires (in the worst case) the communication of $\Omega(q)$ bits. This lower bound applies to randomized protocols with bounded 2-sided error and also to nondeterministic protocols.*

We now present a reduction from 2-party SET DISJOINTNESS to the question of determining whether a system with deterministic, historyless and self-independent reaction functions is convergent. Given an instance of SET-DISJOINTNESS we construct a system with n nodes, each with two actions, as follows (the relation between the parameter q in SET DISJOINTNESS and the number of nodes n is to be specified later). Let the action space of each node be $\{x, y\}$. We now define the reaction functions of the nodes. Consider the possible action profiles of nodes $3, \ldots, n$, *i.e.*, the set $\{x, y\}^{n-2}$. Observe that this set of actions can be regarded as the $(n-2)$-hypercube Q_{n-2}, and thus can be visualized as the graph whose vertices are indexed by the binary $(n-2)$-tuples and such that two vertices are adjacent if and only if the corresponding $(n-2)$-tuples differ in exactly one coordinate.

Definition D.3. (chordless paths, snakes) A *chordless path* in a hypercube Q_n is a path $P = (v_0, \ldots, v_w)$ such that for each v_i, v_j on P, if v_i and v_j are neighbors in Q_n then $v_j \in \{v_{i-1}, v_{i+1}\}$. A *snake* in a hypercube is a simple chordless cycle.

The following result is due to Abbot and Katchalski [1].

Theorem D.4. *[1] Let $t \in \mathbb{N}$ be sufficiently large. Then, the size $|S|$ of a maximal snake in the z-hypercube Q_z is at least $\lambda \times 2^z$ for some $\lambda \geq 0.3$.*

Hence, the size of a maximal snake in the Q_{n-2} hypercube is $\Omega(2^n)$. Let S be a maximal snake in $\{x, y\}^{n-2}$. We now show our reduction from SET DISJOINTNESS with $q = |S|$. We identify each element $j \in \{1 \ldots, q\}$ with a unique vertex $v_j \in S$. W.l.o.g we can assume that x^{n-2} is on S (otherwise we can rename nodes' actions to achieve this). For ease of exposition we also assume that y^{n-2} is not on S (getting rid of this assumption is easy). Nodes' reaction functions are as follows.

- Node 1: If $v_j = (v_{j,1}, \ldots, v_{j,n-2}) \in S$ is a vertex that corresponds to an element $j \in E^A$, then $f_1(y, v_{j,1}, \ldots, v_{j,n-2}) = x$; otherwise, f_1 outputs y.
- Node 2: If $v_j = (v_{j,1}, \ldots, v_{j,n-2}) \in S$ is a vertex that corresponds to an element $j \in E^B$, then $f_2(y, v_{j,1}, \ldots, v_{j,n-2}) = x$; otherwise, f_2 outputs y.
- Node $i \in \{3, \ldots, n\}$: if the actions of nodes 1 and 2 are *not* both x then the action y is chosen; otherwise, f_i only depends on actions of nodes in $\{3, \ldots, n\}$ and therefore to describe f_i it suffices to orient the edges of the hypercube Q_{n-2} (an edge from one vertex to another vertex that differs from it in the ith coordinate determines the outcome of f_i for both). This is done as follows: orient the edges in S so as to create a cycle (in one of two possible ways); orient edges between vertices not in S to vertices in S towards the vertices in S; orient all other edges arbitrarily.

Observation D.5. *y^n is the unique stable state of the system.*

In our reduction Alice simulates node 1 (whose reaction function is based on E^A), Bob simulates node 1 (whose reaction function is based on E^B), and one of the two parties simulates all other nodes (whose reaction functions are not based on neither E^A nor E^B). The theorem now follows from the combination of the following claims:

Claim D.6. *In an oscillation it must be that there are infinitely many time steps in which both node 1 and 2's actions are x.*

Proof. By contradiction. Consider the case that from some moment forth it is never the case that both node 1 and 2's actions are x. Observe that from that time onwards the nodes $3, \ldots, n$ will always choose the action y. Hence, after some time has passed the actions of all nodes in $\{3, \ldots, n\}$ will be y. Observe that whenever nodes 1 and 2 are activated thereafter they shall choose the action y and so we have convergence to the stable state y^n. □

Claim D.7. *The system is not convergent iff $E^A \cap E^B \neq \emptyset$.*

Proof. We know (Claim D.6) that if there is an oscillation then there are infinitely many time steps in which both node 1 and 2's actions are x. We argue that this implies that there must be infinitely many time steps in which both nodes select action x *simultaneously*. Indeed, recall that node 1 only chooses action x if node 2's action is y, and vice versa, and so if both nodes never choose x simultaneously, then it is never the case that both nodes' actions are x at the same time step (a contradiction). Now, when is it possible for both 1 and 2 to choose x at the same time? Observe that this can only be if the actions of the nodes in $\{3, \ldots, n\}$ constitute an element that is in both E^A and E^B. Hence, $E^A \cap E^B \neq \emptyset$. □

□

D.2 Computational Complexity

The above communication complexity hardness result required the representation of the reaction functions to (potentially) be exponentially long. What if the reaction functions can be succinctly described? We now present a strong computational complexity hardness result for the case that each reaction function f_i is deterministic and historyless, and is given explicitly in the form of a boolean circuit (for each $a \in A$ the circuit outputs $f_i(a)$).

Theorem 7.2 1. Determining if a system with n nodes, each with a deterministic and historyless reaction function, is convergent is PSPACE-complete.

Proof. Our proof is based on the proof of Fabrikant and Papadimitriou [9] that BGP safety is PSPACE-complete. Importantly, the result in [9] does not imply Theorem 7.2 since [9] only considers dynamics in which nodes are activated one at a time. We present a reduction from the following problem.

STRING NONTERMINATION: The input is a function $g : \Gamma^t \to \Gamma \cup \{halt\}$, for some alphabet Γ, given in the form of a boolean circuit. The objective is to determine whether there exists an initial string $T = (T_0, \ldots, T_{t-1}) \in \Gamma^t$ such that the following procedure *does not* halt.

1. i:=0

2. While $g(T) \neq halt$ do

   - $T_i := g(T)$
   - $i := (i+1)$ *modulu* t

STRING NONTERMINATION is closely related to STRING HALTING from [9] and is also PSPACE-complete. We now present a reduction from STRING NONTERMINATION to the question of determining whether a system with deterministic and historyless reaction functions is convergent.

We construct a system with $n = t + 1$ nodes. The node set is divided into t "*index nodes*" $0, \ldots, t-1$ and a single "*counter node*" x. The action space of each index node is $\Gamma \cup \{halt\}$ and the action space of the counter node is $\{0, \ldots, t-1\} \times (\Gamma \cup \{halt\})$. Let $a = (a_0, \ldots, a_{t-1}, a_x)$ be an action profile of the nodes, where $a_x = (j, \gamma)$ is the action of the counter node. We now define the deterministic and historyless reaction functions of the nodes:

- The reaction function of index node $i \in \{0, \ldots, t-1\}$, f_i: if $\gamma = halt$, then $f_i(a) = halt$; otherwise, if $j = i$, and $a_j \neq \gamma$, then $f_i(a) = \gamma$; otherwise, $f_i(a) = a_i$.
- The reaction function of the counter node, f_x: if $\gamma = halt$, then $f_x(a) = a_x$; if $a_j = \gamma$, then $f_i(a) = ((j+1)$ *modulu* $t, g(a_0, \ldots, a_{t-1})$; otherwise $f_i(a) = a_x$.

The theorem now follows from the following claims that, in turn, follow from our construction:

Claim D.8. *$(halt, \ldots, halt)$ is the unique stable state of the system.*

Proof. Observe that $(halt, \ldots, halt)$ is indeed a stable state of the system. The uniqueness of this stable state is proven via a simple case-by-case analysis. □

Claim D.9. *If there exists an initial string $T = (T_0, \ldots, T_{t-1})$ for which the procedure does not terminate then there exists an initial state from which the system does not converge to the stable state $(halt, \ldots, halt)$ regardless of the schedule chosen.*

Proof. Consider the evolution of the system from the initial state in which the action of index node i is T_i and the action of the counter node is $(0, g(T))$. □

Claim D.10. *If there does not exist an initial string T for which the procedure does not terminate then the system is convergent.*

Proof. Observe that if there is an initial state $a = (a_0, \ldots, a_{t-1}, a_x)$ and a fair schedule for which the system does not converge to the unique stable state then the procedure does not halt for the initial string $T = (a_0, \ldots, a_{t-1})$. □

□

Proving the above PSPACE-completeness result for the case self-independent reaction functions seems challenging.

Problem D.11. *Prove that determining if a system with n nodes, each with a deterministic self-independent and historyless reaction function, is convergent is PSPACE-complete.*

E Some Basic Observations Regarding No-Regret Dynamics

Regret minimization is fundamental to learning theory. The basic setting is as follows. There is a space of m actions (*e.g.*, possible routes to work), which we identify with the set $[m] = \{1, \ldots, m\}$. In each time step $t \in \{1, \ldots\}$, an adversary selects a profit function $p_t : [m] \to [0, 1]$ (*e.g.*, how fast traffic is flowing along each route), and the (randomized) algorithm chooses a distribution D_t over the elements in $[m]$. When choosing D_t the algorithm can only base its decision on the profit functions $p_1, \ldots, p_{t-1}$, and not on p_t (that is revealed only after the algorithm makes its decision). The algorithm's gain at time t is $g_t = \Sigma_{j \in [m]} D_t(j) p_t(j)$, and its accumulated gain at time t is $\Sigma_{i=1}^{t} g_t$. Regret analysis is useful for designing adaptive algorithms that fair well in such uncertain environments. The motivation behind regret analysis is ensuring that, over time, the algorithm performs at least as well in retrospect as some alternative "simple" algorithm.

We now informally present the three main notions of regret (see [3] for a thorough explanation): (1) *External regret* compares the algorithm's performance to that of simple algorithms that select the exact same action in each time step (*e.g.*, "you should have always taken Broadway, and never chosen other routes"). (2) *Internal regret* and *swap regret* analysis compares the gain from the sequence of actions actually chosen to that derived from replacing every occurrence of an action i with another action j (*e.g.*, "every time you chose Broadway you should have taken 7th Avenue instead). While internal regret analysis allows only *one* action to be replaced by another, swap regret analysis considers all mappings from $[m]$ to $[m]$. The algorithm has *no (external/internal/swap) regret* if the gap between the algorithm's gain and the gain from the best alternative policy allowed vanishes with time.

Regret minimization has strong connections to game-theoretic solution concepts. If each player in a repeated game executes a no-regret algorithm when selecting strategies, then convergence to an equilibrium is guaranteed in a variety of interesting contexts. The notion of convergence, and the kind of equilibrium reached, vary, and are dependent on the restrictions imposed on the game and on the type of regret being minimized (*e.g.*, in zero-sum games, no-external-regret algorithms are guaranteed to approach or exceed the minimax value of the game; in general games, if all players minimize swap regret, then the empirical distribution of joint players' actions converges to a correlated equilibrium, *etc.*). (See [3] and references therein). Importantly, these results are all proven within a model of interaction in which each player selects a strategy in each and every time step.

We make the following simple observation. Consider a model in which the adversary not only chooses the profit functions but also has the power not to allow the algorithm to select a new distribution over actions in some time steps. That is, the adversary also selects a schedule σ such that $\forall t \in \mathbb{N}_+$, $\sigma(t) \in \{0, 1\}$, where 0 and 1 indicate whether the algorithm is not activated, or activated, respectively. We restrict the schedule to be r-fair, in the sense that the schedule chosen must be such that the algorithm is activated at least once in every r consecutive time steps. If the algorithm is activated at time t and not activated again until time $t + \beta$ then it holds that $\forall s \in \{t + 1, \ldots, t + \beta - 1\}$, $D_s = D_t$ (the algorithm cannot change its probability distribution over actions while not activated). We observe that if an algorithm has no regret in the above setting (for all three notions of regret), then it has no regret in this setting as well. To see this, simply observe that if we regard each batch of time steps in which the algorithms is not activated as one "meta time step", then this new setting is equivalent to the traditional setting (with $p_t : [m] \to [0, r]$ for all $t \in \mathbb{N}_+$).

This observation, while simple, is not uninteresting, as it implies that *all* regret-based results for

repeated games continue to hold even if players' order of activation is *asynchronous* (see Section 3 for a formal exposition of asynchronous interaction), so long as the schedule of player activations is r-fair for some $r \in N_+$. We mention two implications of this observation.

Observation E.1. *When all players in a zero-sum game use no-external-regret algorithms then approaching or exceeding the minimax value of the game is guaranteed.*

Observation E.2. *When all players in a (general) game use no-swap-regret algorithms the empirical distribution of joint players' actions converges to a correlated equilibrium of the game.*

Problem E.3. *Give examples of repeated games for which there exists a schedule of player activations that is not r-fair for any $r \in N_+$ for which regret-minimizing dynamics do not converge to an equilibrium (for different notions of regret/convergence/equilibria).*

Problem E.4. *When is convergence of no-regret dynamics to an equilibrium guaranteed (for different notions of regret/convergence/equilibria) for all r-fair schedules for non-fixed r's, that is, if when r is a function of t?*

F An Axiomatic Approach

We now use (a slight variation of) the framework of Taubenfeld, which he used to study resilient consensus protocols [31], to prove Thm. 4.1. We first (Sec. F.2) define *runs* as sequences of *events*; unlike Taubenfeld, we allow infinite runs. A protocol is then a collection of runs (which must satisfy some natural conditions like closure under taking prefixes). We then define colorings of runs (which correspond to outcomes that can be reached by extending a run in various ways) and define the IoD property.

The proof of Thm. 4.1 proceeds in two steps. First, we show that *any* protocol that satisfies IoD has some (fair, as formalized below), non-terminating activation sequence. We then show that protocols that satisfy the hypotheses of Thm. 4.1 also satisfy IoD.

F.1 Proof Sketch

Proof Sketch. The proof follows the axiomatic approach of Taubenfeld [31] in defining asynchronous protocols in which states are colored by sets of colors; the set of colors assigned to a state must be a superset of the set of colors assigned to any state that is reachable (in the protocol) from it. We then show that any such protocol that satisfies a certain pair of properties (which we call *Independence of Decisions* or *IoD*) and that has a polychromatic state must have a non-terminating fair run in which all states are polychromatic.

For protocols with 1-recall, self-independence, and stationarity, we consider (in order to reach a contradiction) protocols that are guaranteed to converge. Each starting state is thus guaranteed to reach only stable states; we then color each state according to the outcomes that are reachable from that state. We show that, under this coloring, such protocols satisfy IoD and that, as in consensus protocols, the existence of multiple stable states implies the existence of a polychromatic state. The non-terminating, polychromatic, fair run that is guaranteed to exist is, in the context, exactly the non-convergent protocol run claimed by the theorem statement. We then show that this may be extended to non-stationary protocols with k-recall (for $k > 1$). □

F.2 Events, Runs, and Protocols

Events are the atomic actions that are used to build runs of a protocol. Each event is associated with one or more principals; these should be thought of as the principals who might be affected by the event (*e.g.*, as sender or receiver of a message), with the other principals unable to see the event. We start with the following definition.

Definition F.1 (Events and runs)**.** There is a set E whose elements are called *events*; we assume a finite set of possible events (although there will be no restrictions on how often any event may occur). There is a set $\mathcal{P}$ of *principals*; each event has an associated set $S \subseteq \mathcal{P}$, and if S is the set associated to $e \in E$, we will write e_S.

There is a set $\mathcal{R}$ whose elements are called *runs*; each run is a (possibly infinite) sequence of events. We say that event e is *enabled* at run $\mathbf{x}$ if the concatenation $\langle \mathbf{x}; e \rangle$ (*i.e.*, the sequence of events that is $\mathbf{x}$ followed by the single event e) is also a run. (We will require that $\mathcal{R}$ be prefix-closed in the protocols we consider below.)

The definition of a protocol will also make use of a couple types of relationship between runs; our intuition for these relationships continues to view e_P as meaning that event e affects the set P of principals. From this intuitive perspective, two runs are equivalent with respect to a set S of principals exactly when their respective subsequences that affect the principals in S are identical. We also say that one run includes another whenever, from the perspective of every principal (*i.e.*, restricting to the events that affect that principal), the included run is a prefix of the including run. Note that this does not mean that the sequence of events in the included run is a prefix of the sequence of events in the including run—events that affect disjoint sets of principals can be reordered without affecting the inclusion relationship.

Definition F.2 (Run equivalence and inclusion)**.** For a run $\mathbf{x}$ and $S \subseteq \mathcal{P}$, we let $\mathbf{x}_S$ denote the subsequence (preserving order and multiplicity) of events e_P in $\mathbf{x}$ for which $P \cap S \neq \emptyset$. We say that $\mathbf{x}$ and $\mathbf{y}$ are *equivalent with respect to* S, and we write $\mathbf{x}[S]\mathbf{y}$, if $\mathbf{x}_S = \mathbf{y}_S$. We say that $\mathbf{y}$ *includes* $\mathbf{x}$ if for every node i, the restriction of $\mathbf{x}$ to those events e_P with $i \in P$ is a prefix of the restriction of $\mathbf{y}$ to such events.

Our definitions of $\mathbf{x}_S$ and $\mathbf{x}[S]\mathbf{y}$ generalize definitions given by Taubenfeld [31] for $|S| = 1$—allowing us to consider events that are seen by multiple principals—but other than this and the allowance of infinite runs, the definitions we use in this section are the ones he used. Importantly, however, we do not use the resilience property that Taubenfeld used.

Finally, we have the formal definition of an asynchronous protocol. This is a collection of runs that is closed under taking prefixes and only allows for finitely many (possibly 0) choices of a next event to extend the run. It also satisfies the property (P_2 below) that, if a run can be extended by an event that affects exactly the set S of principals, then any run that includes this run and that is equivalent to the first run with respect to S (so that only principals not in S see events that they don't see in the first run) can also be extended by the same event.

Definition F.3 (Asynchronous protocol)**.** An *asynchronous protocol* (or just a *protocol*) is a collection of runs that satisfies the following three conditions.

P_1 Every prefix of a run is a run.

P_2 Let $\langle \mathbf{x}; e_S \rangle$ and $\mathbf{y}$ be runs. If $\mathbf{y}$ includes $\mathbf{x}$, and if $\mathbf{x}[S]\mathbf{y}$, then $\langle \mathbf{y}; e_S \rangle$ is also a run.

P_3 Only finitely many events are enabled at a run.

F.3 Fairness, Coloring, and Decisions

We start by recalling the definition of a fair sequence [31]; as usual, we are concerned with the behavior of fair runs. We also introduce the notion of a fair extension, which we will use to construct fair infinite runs.

Definition F.4 (Fair sequence, fair extension)**.** We define a *fair sequence* to be a sequence of events such that: every finite prefix of the sequence is a run; and, if the sequence is finite, then no event is enabled at the sequence, while if the sequence is infinite, then every event that is enabled at all but finitely many prefixes of the sequence appears infinitely often in the sequence. We define a *fair extension* of a (not necessarily fair) sequence $\mathbf{x}$ to be a finite sequence $e_1, e_2, \ldots, e_k$ of events such that e_1 is enabled at $\mathbf{x}$, e_2 is enabled at $\langle \mathbf{x}; e_1 \rangle$, *etc.*

We also assign a set of "colors" to each sequence of events subject to the conditions below. As usual, the colors assigned to a sequence will correspond to the possible protocol outcomes that might be reached by extending the sequence.

Definition F.5 (Asynchronous, C-chromatic protocol)**.** Given a set C (called the set of *colors*), we will assign sets of colors to sequences; this assignment may be a partial function. For a set C, we will say that a protocol is *C-chromatic* if it satisfies the following properties.

C_1 For each $c \in C$, there is a protocol run of color $\{c\}$.

C_2 For each protocol run $\mathbf{x}$ of color $C' \subseteq C$, and for each $c \in C'$, there is an extension of $\mathbf{x}$ that has color $\{c\}$.

C_3 If $\mathbf{y}$ includes $\mathbf{x}$ and $\mathbf{x}$ has color C', then the color of $\mathbf{y}$ is a subset of C'.

We say that a fair sequence is *polychromatic* if the set of colors assigned to it has more than one element.

Finally, a C-chromatic protocol is called a *decision protocol* if it also satisfies the following property:

D Every fair sequence has a finite monochromatic prefix, *i.e.*, a prefix whose color is $\{c\}$ for some $c \in C$.

F.4 Independence of Decisions (IoD)

We turn now to the key (two-part) condition that we use to prove our impossibility results.

Definition F.6 (Independence of Decisions (IoD))**.** A protocol satisfies *Independence of Decisions* (IoD) if, whenever

- a run $\mathbf{x}$ is polychromatic and
- there is some event e is enabled at $\mathbf{x}$ and $\langle \mathbf{x}; e \rangle$ is monochromatic of color $\{c\}$,

then

1. for every $e' \neq e$ that is enabled at $\mathbf{x}$, the color of $\langle \mathbf{x}; e' \rangle$ contains c, and
2. for every $e' \neq e$ that is enabled at $\mathbf{x}$, if $\langle \langle \mathbf{x}; e' \rangle ; e \rangle$ is monochromatic, then its color is also $\{c\}$.

Figure 1 illustrates the two conditions that form IoD. Both parts of the figure include the polychromatic run $\mathbf{x}$ that can be extended to $\langle \mathbf{x}; e \rangle$ with monochromatic color $\{c\}$; the color of $\mathbf{x}$ necessarily includes c. The left part of the figure illustrates condition 1, and the right part of the figure illustrates condition 2. The dashed arrow indicates a sequence of possibly many events,

Figure 1: Illustration of the two conditions of IoD.

while the solid arrows indicate single events. The labels on a node in the figure indicate what is assumed/required about the set that colors the node.

Condition 1 essentially says that, if an event e decides the outcome of the protocol, then no other event can rule out the outcome that e produced. The name "Independence of Decisions" derives from condition 2, which essentially says that, if event e decides the outcome of the protocol both before and after event e', then the decision that is made is independent of whether e' happens immediately before or after e.

In working with IoD-satisfying protocols, the following lemma will be useful.

Lemma F.7. *If* IoD *holds, then for any two events e and e' that are enabled at a run $\mathbf{x}$, if both $\langle \mathbf{x}; e\rangle$ and $\langle \mathbf{x}; e'\rangle$ are monochromatic, then those colors are the same.*

Proof. By IoD, the color of $\langle \mathbf{x}; e'\rangle$ must contain the color of $\langle \mathbf{x}; e\rangle$, and both of these sets are singletons. □

F.5 IoD-Satisfying Protocols Don't Always Converge

To show that IoD-satisfying protocols don't always converge, we proceed in two steps: first, we show (Lemma F.8) that a polychromatic sequence can be fairly extended (in the sense of ...) to another polychromatic sequence; second, we use that lemma to show (Thm. F.9) ....

Lemma F.8 (The Fair-Extension Lemma)**.** *In a polychromatic decision protocol that satisfies* IoD*, if a run $\mathbf{x}$ is polychromatic, then $\mathbf{x}$ can be extended by a fair extension to another polychromatic run.*

Proof. Assume that, for some C', there is a run $\mathbf{x}$ of color C' that cannot be fairly extended to another polychromatic run. Because $|C'| > 1$, there must be some event that is enabled at $\mathbf{x}$; if not, we would contradict D. Figure 2 illustrates this (and the arguments in the rest of the proof below).

Consider the extensions of $\mathbf{x}$ that use as many distinct events as possible and that are polychromatic, and pick one of these $\mathbf{y}$ that minimizes the number of events that are enabled at every prefix of $\mathbf{y}$ (after $\mathbf{x}$ has already been executed) but that do not appear in $\mathbf{y}$. If $\mathbf{y}$ contains no events (illustrated in the top left of Fig. 2), then every event e that is enabled at $\mathbf{x}$ is such that $\langle x; e\rangle$ is monochromatic. By Lemma F.7, these singletons must all be the same color $\{c\}$; however, this

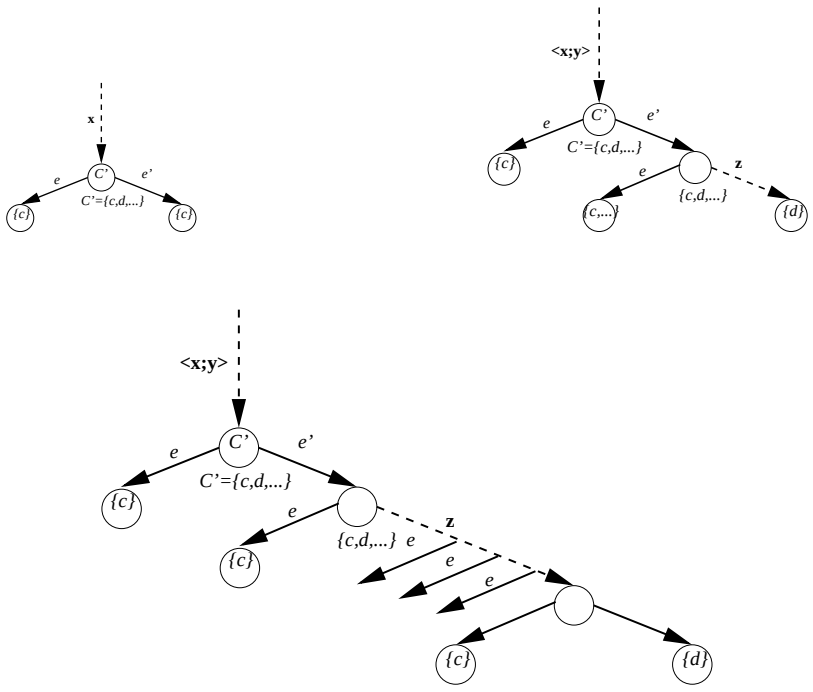

Figure 2: Illustration of proof of Lem. F.8.

means that for $c' \in C' \setminus \{c\} \neq \emptyset$, $\mathbf{x}$ does not have any extensions whose color is c', contradicting C_2.

If $\mathbf{y}$ contains one or more events (illustrated in the top right and bottom of Fig. 2), then (because it is not a fair extension of $\mathbf{x}$) there is at least one event e that is enabled everywhere in the extension, including at $\langle \mathbf{x}; \mathbf{y} \rangle$, but that does not appear anywhere in $\mathbf{y}$. Because $\mathbf{y}$ was chosen instead of $\langle \mathbf{y}; e \rangle$ (or another extension with the same number of distinct events), the color of $\langle \langle \mathbf{x}; \mathbf{y} \rangle ; e \rangle$ must be a singleton $\{c\}$. Because $\langle \mathbf{x}; \mathbf{y} \rangle$ is polychromatic, it has some extension $\mathbf{z}$ that is (eventually) monochromatic with color $\{d\} \neq \{c\}$; let e' be the first event in this extension. Because IoD is satisfied, the color of $\langle \langle \mathbf{x}; \mathbf{y} \rangle ; e' \rangle$ also contains c and is thus polychromatic. The event e is again enabled here (else $\langle \langle \mathbf{x}; \mathbf{y} \rangle ; e' \rangle$ would have been chosen instead of $\mathbf{y}$). If $\langle \langle \langle \mathbf{x}; \mathbf{y} \rangle ; e' \rangle ; e \rangle$ is not monochromatic (top right of Fig. 2), then it is a polychromatic extension of $\mathbf{x}$ that uses more distinct events than does $\mathbf{y}$, a contradiction. If $\langle \langle \langle \mathbf{x}; \mathbf{y} \rangle ; e' \rangle ; e \rangle$ is monochromatic (bottom of Fig. 2), then by IoD it has color $\{c\}$. We may then inductively move along the extension $\mathbf{z}$; after each additional event from $\mathbf{z}$ is appended to the run, the resulting run is polychromatic (its color set must include d, but if it is monochromatic it must have color $\{c\}$) and again enables e (by our choice of $\mathbf{y}$). Again by our choice of $\mathbf{y}$, appending e to this run must produce a monochromatic run, which (by IoD) must have color $\{c\}$. Proceeding along $\mathbf{z}$, we must then eventually reach a polychromatic run at which e is enabled (and produces a monochromatic run of color $\{c\}$) and which also enables a different event that yields a monochromatic run of color $\{d\}$. This contradicts Lem. F.7. □

Theorem F.9. *Any* IoD*-satisfying asynchronous protocol with a polychromatic initial state has a fair sequence that starts at this initial state and never reaches a decision,* i.e.*, it has a fair sequence*

that does not have a monochromatic prefix.

Proof. Start with the empty (polychromatic) run and iteratively apply the fair-extension lemma to obtain an infinite polychromatic sequence. If an event e is enabled at all but finitely many prefixes in this sequence, then in all but finitely many of the fair extensions, e is enabled at every step of the extension. Because these extensions are fair (in the sense of Def. F.4), e is activated in each of these (infinitely many) extensions and so appears infinitely often in the sequence, which is thus fair. □

F.6 1-Recall, Stationary, Self-Independent Protocols Need Not Converge

We first recall the statement of Thm. 4.1. We then show that 1-recall, historyless protocols satisfy IoD when colored as in Def. F.10. Theorem F.9 then implies that such protocols do not always converge; it immediately follows that this also applies to bounded-recall (and not just 1-recall) protocols.

Theorem 4.1. *If each node i has bounded recall, and each reaction function f_i is self-independent and stationary, then the existence of two stable states implies that the computational network is not safe.*

Definition F.10 (Stable coloring)**.** In a protocol defined as in Sec. 3, the *stable coloring* of protocol states is the coloring that has a distinct color for each stable state and that colors each state in a run with the set of colors corresponding to the stable states that are reachable from that state.

We model the dynamics of a 1-recall, historyless protocol as follows. There are two types of actions: the application of nodes' reaction functions, where e_i is the action of node i acting as dictated by f_i, and a "reveal" action W. The nodes scheduled to react in the first timestep do so sequentially, but these actions are not yet visible to the other nodes (so that nodes after the first one in the sequence are still reacting to the initial state and not to the actions performed earlier in the sequence). Once all the scheduled nodes have reacted, the W action is performed; this reveals the newly performed actions to all the other nodes in the network. The nodes that are scheduled to react at the next timestep then act in sequence, followed by another W action, and so on. This converts the simultaneous-action model of Sec. 3 to one in which actions are performed sequentially; we will use this "act-and-tell" model in the rest of the proof. We note that all actions are enabled at every step (so that, *e.g.*, e_i can be taken multiple times between W actions; however, this is indistinguishable from a single e_i action because the extra occurrences are not seen by other nodes, and they do not affect i's actions, which are governed by a historyless reaction function).

Once we cast the dynamics of 1-recall, historyless protocols in the act-and-tell model, the following lemma will be useful.

Lemma F.11 (Color equalities)**.** *In a 1-recall, historyless protocol (in the act-and-tell model):*

1. *For every run pair of runs $\mathbf{x}, \mathbf{y}$ and every $i \in [n]$, the color of $\langle\langle \mathbf{x}; e_i W e_i W \rangle ; \mathbf{y}\rangle$ is the same as the color of $\langle\langle \mathbf{x}; W e_i W \rangle ; \mathbf{y}\rangle$.*
2. *For every run pair of runs $\mathbf{x}, \mathbf{y}$ and every $i, j \in [n]$, the color of $\langle\langle \mathbf{x}; e_i e_j \rangle ; \mathbf{y}\rangle$ is the same as the color of $\langle\langle \mathbf{x}; e_j e_i \rangle ; \mathbf{y}\rangle$.*

Informally, the first color equality says that, if all updates are announced and then i activates and then all updates are revealed again (i's new output being the only new one), it makes no difference

whether or not i was activated immediately before the first reveal action. The second color equality says that, as long as there are no intervening reveal event, the order in which nodes compute their outputs does not matter (because they do not have access to their neighbors' new outputs until the reveal event).

Proof. For the first color equality, because the protocol is self-independent, the first occurrence of e_i (after $\mathbf{x}$) in $\langle\langle \mathbf{x}; e_i W e_i W\rangle ; \mathbf{y}\rangle$ does not affect the second occurrence of e_i. Because the protocol has 1-recall, the later events (in $\mathbf{y}$) are also unaffected.

The second color equality is immediate from the definition of the act-and-tell model. □

Lemma F.12. *If a protocol is* 1*-recall and historyless, then the protocol (with the stable coloring) satisfies* IoD.

Proof. Color each state in the protocol's runs according to the stable states that can be reached from it. Assume $\mathbf{x}$ is a polychromatic run (with color C') and that some event e is such that $\langle \mathbf{x}; e\rangle$ is monochromatic (with color $\{c\}$). Let e' be another event (recall that all events are always enabled). If e and e' are two distinct node events e_i and e_j ($i \neq j$), respectively, then the color of $\langle\langle \mathbf{x}; e_j\rangle ; e_i\rangle$ is the color of $\langle\langle \mathbf{x}; e_i\rangle ; e_j\rangle$ and thus the (monochromatic) color of $\langle \mathbf{x}; e_i\rangle$, *i.e.*, $\{c\}$. If e and e' are both W or are the same node event e_i, then the claim is trivial.

If $e = e_i$ and $e' = W$ (as illustrated in the left of Fig. 3), then we may extend $\langle \mathbf{x}; e_i\rangle$ by We_iW to obtain a run whose color is again $\{c\}$. By the second color equality, this is also the color of the extension of $\langle \mathbf{x}; W\rangle$ by e_iW, so the color of $\langle \mathbf{x}; W\rangle$ contains c and if the extension of $\langle \mathbf{x}; W\rangle$ by e_i is monochromatic, its color must be $\{c\}$ as well. If, on the other hand, $e = W$ and $e' = e_i$ (as illustrated in the right of Fig. 3), we may extend $\langle \mathbf{x}; W\rangle$ by e_iW and $\langle \mathbf{x}; e_i\rangle$ by We_iW to obtain runs of color $\{c\}$; so the color of $\langle \mathbf{x}; e_i\rangle$ must contain c and, arguing as before, if the intermediate extension $\langle\langle \mathbf{x}; e_i\rangle ; W\rangle$ is monochromatic, its color must also be $\{c\}$. □

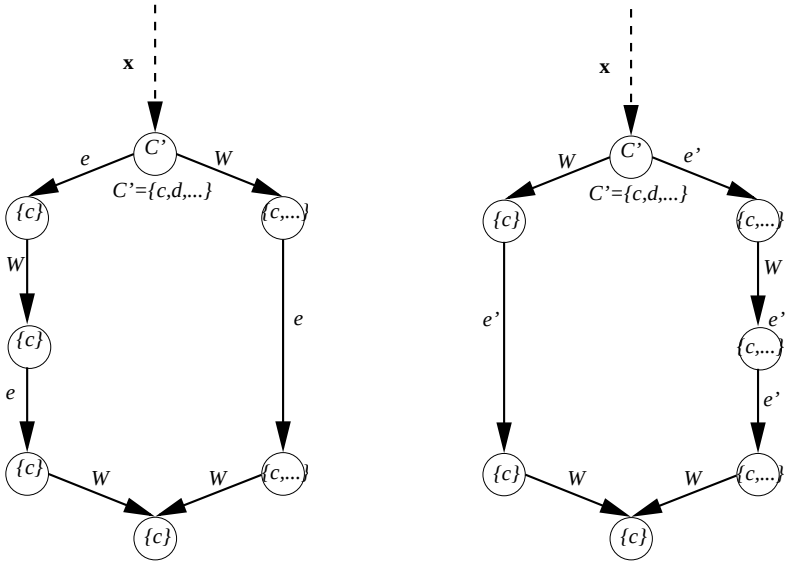

Figure 3: Illustrations of the arguments in the proof of Lem. F.12.

Lemma F.13. *If a* 1*-recall, historyless computation that always converges can, for different starting states, converge to different stable states then there is some input from which the computation can reach multiple stable states. In particular, under the stable coloring, there is a polychromatic state.*

Proof. Assume there are (under the stable coloring) two different monochromatic input states for the computation, that the inputs differ only at one node v, and that the computation always converges (*i.e.*, for every fair schedule) on both input states. Consider a fair schedule that activates v first and then proceeds arbitrarily. Because the inputs to v's reaction function are the same in each case, after the first step in each computation, the resulting two networks have the same node states. This means that the computations will subsequently unfold in the same way, in particular producing identical outputs.

If a historyless computation that always converges can produce two different outputs, then iterated application of the above argument leads to a contradiction unless there is a polychromatic initial state. □

Proof of 1*-recall, stationary part of Thm. 4.1.* Consider a protocol with 1-recall, self independence, and stationarity, and that has two different stable states. If there is some non-convergent run of the protocol, then the network is not safe (as claimed). Now assume that all runs converge; we will show that this leads to a contradiction. Color all states in the protocol's runs according to the stable coloring (Def. F.10). Lemma F.13 implies that there is a polychromatic state. Because, by Lem. F.12, the IoD is satisfied, we may apply Thm. F.9. In this context (with the stable coloring), this implies that there is an infinite run in which every state can reach at least two stable states; in particular, the run does not converge. □

F.7 Extension to Non-stationary Protocols

We may extend our results to non-stationary protocols as well.

Theorem F.14. *If each node i has* 1*-recall, the action spaces are all finite, and each reaction function f_i is self-independent but not necessarily stationary, then the existence of two stable states implies that the computational network is not safe.*

Proof. In this context, a stable state is a vector of actions and a time t such that, after t, the action vector is a fixed point of the reaction functions. Let T be the largest such t over all the (finitely many) stable states (and ensure that T is at least k for generalizing to k-recall). Assume that the protocol is in fact safe; this means that, under the stable coloring, every state gets at least one color. If there are only monochromatic states, consider the states at time T; we view two of these states as adjacent if they differ only in the action (or action history for the generalization to k-recall) of one node. Because the protocol is self-independent, that node may be activated (k times if necessary) to produce the same state. In particular, this means that adjacent states must have the same monochromatic color. Because (among he states at time T) there is a path (following state adjacencies) from any one state to any other, only one stable state is possible, contradicting the hypotheses of the theorem.

Considering the proof of Lem. F.12, we see that the number of timesteps required to traverse each of the subfigures in Fig. 3 does not depend on which path (left or right) through the subfigure we take. In particular, this means that the reaction functions are not affected by the choice of path. Furthermore, the non-W actions in each subfigure only involve a single node i; the final

action performed by i along each path occurs after one W action has been performed (after $\mathbf{x}$), so these final actions are the same (because the timesteps at which they occur are the same, as are the actions of all the other nodes in the network). □

F.8 Extension to Bounded-Recall Protocols

If we allow k-recall for $k > 1$, we must make a few straightforward adjustments to the proofs above. Generalizing the argument used in the proof of the color equalities (Lem. F.11), we may prove an analogue of these for k-recall; in particular, we replace the first color equality by an equality between the colors of $\langle\langle \mathbf{x}; e_i W(e_i W)^k \rangle ; \mathbf{y}\rangle$ and $\langle\langle \mathbf{x}; W(e_i W)^k \rangle ; \mathbf{y}\rangle$. This leads to the analogue of Lem. F.12 for bounded-recall protocols; as in Lem. F.12, the two possible paths through each subfigure (in the k-recall analogue of Fig. 3) require the same number of timesteps, so non-stationarity is not a problem.

Considering adjacent states as those that differ only in the actions of one node (at some point in its depth-k history), we may construct a path from any monochromatic initial state to any other such state. Because the one node that differs between two adjacent states may be (fairly) activated k times to start the computation, two monochromatic adjacent states must have the same color; as in the 1-recall case, the existence of two stable states thus implies the existence of a polychromatic state.

G Implications for Resilient Decision Protocols

The consensus problem is fundamental to distributed computing research. We give a brief description of it here, and we refer the reader to [31] for a detailed explanation of the model. We then show how to apply our general result to this setting. This allows us to show that the impossibility result in [12], which shows that no there is no protocol that solves the consensus problem, can be obtained as a corollary of Thm. F.9.

G.1 The Consensus Problem

Processes and consensus. There are $N \geq 2$ *processes* $1, \ldots, N$, each process i with an initial value $x_i \in \{0, 1\}$. The processes communicate with each other via *messages*. The objective is for all *non-faulty* processes to eventually agree on some value $x \in \{0, 1\}$, such that $x = x_i$ for some $i \in [N]$ (that is, the value that has been decided must match the initial value of some process). No computational limitations whatsoever are imposed on the processes. The difficulty in reaching an agreement (consensus) lies elsewhere: the network is asynchronous, and so there is no upper bound on the length of time processes may take to receive, process and respond to an incoming message. Intuitively, it is therefore impossible to tell whether a process has failed, or is simply taking a long time.

Messages and the message buffer. Messages are pairs of the form (p, m), where p is the process the message is intended for, and m is the contents of the message. Messages are stored in an abstract data structure called the *message buffer*. The message buffer is a multiset of messages, *i.e.*, more than one of any pair (p, m) is allowed, and supports two operations: (1) *send(p,m)*: places a message in the message buffer. (2) *receive(p)*: returns a message for processor p (and removes it

from the message buffer) or the special value, that has no effects. If there are several messages for p in the message buffer then receive(p) returns one of them at random.

Configurations and system evolution. A configuration is defined by the following two factors: (1) the internal state of all of the processors (the current step in the protocol that they are executing, the contents of their memory), and (2) the contents of the message buffer. The system moves from one configuration to the next by a step which consists of a process p performing *receive(p)* and moving to another internal state. Therefore, the only way that the system state may evolve is by some processor receiving a message (or null) from the message buffer. Each step is therefore uniquely defined by the message that is received (possibly) and the process that received it.

Executions and failures. From any initial starting state of the system, defined by the initial values of the processes, there are many different possible ways for the system to evolve (as the *receive(p)* operation is non-deterministic). We say that a *protocol* solves consensus if the objective is achieved for *every* possible execution. Processes are allowed to fail according to the fail-stop model, that is, processes that fail do so by ceasing to work correctly. Hence, in each execution, non-faulty processes participate in infinitely many steps (presumably eventually just receiving once the algorithm has finished its work), while processes that stop participating in an execution at some point are considered faulty. We are concerned with the handling of (at most) *a single* faulty process. Hence, an execution is *admissible* if at most one process is faulty.

G.2 Impossibility of Resilient Consensus

We now show how this fits into the formal framework of Ap. F. The events are (as in [12]) messages annotated with the intended recipient (*e.g.*, m_i). In addition to the axioms of Ap. F, we also assume that the protocol satisfies the following resiliency property, which we adapt from Taubenfeld [31]; we call such a protocol a *resilient consensus protocol*. (Intuitively, this property ensures that if node i fails, the other nodes will still reach a decision.)

Res For each run $\mathbf{x}$ and node i, there is a monochromatic run $\mathbf{y}$ that extends $\mathbf{x}$ such that $\mathbf{x}\,[i]\,\mathbf{y}$.

We show that resilient consensus protocols satisfy IoD. Unsurprisingly, the proof draws on ideas of Fischer, Lynch, and Paterson.

Lemma G.1. *Resilient consensus protocols satisfy* IoD.

Proof. Assume $\mathbf{x}$ is a polychromatic run of a resilient consensus protocol and that $\langle \mathbf{x}; m_i \rangle$ is monochromatic (of color $\{c\}$). If $e' = m'_j$ for $j \neq i$, then $e = m_i$ and e' commute (because the messages are processed by different nodes) and the IoD conditions are satisfied. (In particular, $\langle \langle \mathbf{x}; e \rangle ; e' \rangle$ and $\langle \langle \mathbf{x}; e' \rangle ; e \rangle$ both have the same monochromatic color.)

If $e' = m'_i$, then consider a sequence σ from $\mathbf{x}$ that reaches a monochromatic run and that does not involve i (the existence of σ is guaranteed by Res); this is illustrated in Fig. 4. Because σ doesn't involve i, it must commute with e and e'; in particular, the color of the monochromatic run reachable by applying σ to $\langle \mathbf{x}; e \rangle$ is the same as the color of the run $\langle \langle \mathbf{x}; \sigma \rangle ; e \rangle$. Thus σ must produce the same color $\{c\}$ that e does in extending $\mathbf{x}$. On the other hand, we may apply this same argument to e' to see that $\langle \langle \mathbf{x}; e' \rangle ; \sigma \rangle$ must also have the same color as $\langle \mathbf{x}; \sigma \rangle$, so the color of $\langle \mathbf{x}; e' \rangle$ contains the color of $\langle \mathbf{x}; e \rangle$. The remaining question is whether $\langle \langle \mathbf{x}; e' \rangle ; e \rangle$ can be monochromatic of a different color than $\langle \mathbf{x}; e \rangle$. However, the color (if it is monochromatic) of $\langle \langle \langle \mathbf{x}; e' \rangle ; e \rangle ; \sigma \rangle$ must be the same (because σ does not involve i) as the color of $\langle \langle \langle \mathbf{x}; e' \rangle ; \sigma \rangle ; e \rangle$, which we have already established is the color of $\langle \mathbf{x}; e \rangle$; thus, $\langle \langle \mathbf{x}; e' \rangle ; e \rangle$ cannot be monochromatic of a different color. □

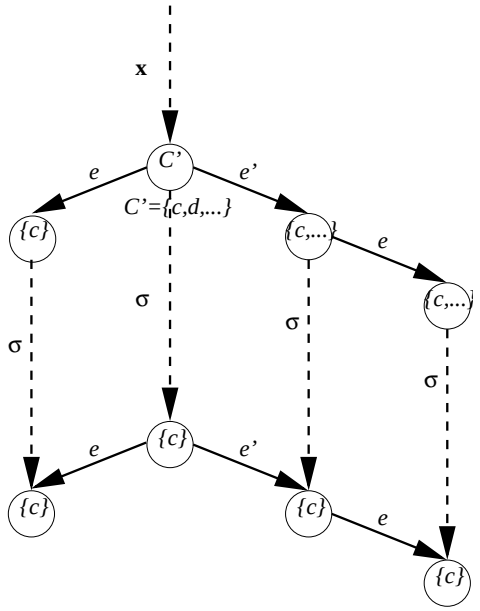

Figure 4: Illustration of argument in the proof of Lem. G.1.

Using Thm. F.9 and the fact that there must be a polychromatic initial configuration for the protocol (because it can reach multiple outcomes, as shown in [12]), we obtain from this lemma the following celebrated result of Fischer, Lynch, and Paterson [12].

Theorem G.2 (Fischer–Lynch–Paterson [12])**.** *There is no always-terminating protocol that solves the consensus problem.*